UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Implementação em Software da Codificação de Canal em ADSL: Um Estudo de Caso Sobre os Efeitos do Ruído Impulsivo na Transmissão de Vídeo

NOME DO AUTOR

Fernanda Regina Smith Neves Corrêa

DM_12/2011

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2011

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NOME DO AUTOR

Fernanda Regina Smith Neves Corrêa

Implementação em Software da Codificação de Canal em ADSL: Um Estudo de Caso Sobre os Efeitos do Ruído Impulsivo na Transmissão de Vídeo

DM_12/2011

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2011

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NOME DO AUTOR

Fernanda Regina Smith Neves Corrêa

Implementação em Software da Codificação de Canal em ADSL: Um Estudo de Caso Sobre os Efeitos do Ruído Impulsivo na Transmissão de Vídeo

Dissertação submetida à Banca Examinadora do Programa de Pós-graduação em Engenharia Elétrica da UFPA para a obtenção do Grau de Mestre em Engenharia Elétrica, ênfase em Telecomunicações.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2011

C824i Corrêa, Fernanda Regina Smith Neves

Implementação em software da codificação de canal em ADSL: um estudo de caso sobre os efeitos do ruído impulsivo na transmissão de vídeo / Fernanda Regina Smith Neves Corrêa; orientador, Evaldo Gonçalves Pelaes.-2011.

Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2011.

1. Modem. 2. Linha digital de assinantes. 3. Cabos de telecomunicação - ruído. I. Orientador. II. Título.

CDD 22. ed. 621.39814

UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Implementação em Software da Codificação de Canal em ADSL: Um Estudo de Caso Sobre os Efeitos do Ruído Impulsivo na Transmissão de Vídeo

AUTOR: FERNANDA REGINA SMITH NEVES CORRÊA

DISSERTAÇÃO DE MESTRADO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADA ADEQUADA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE TELECOMUNICAÇÕES.

APROVADA EM 25/03/2011

BANCA EXAMINADORA:

.....

Prof. Dr. Evaldo Gonçalves Pelaes (ORIENTADOR - UFPA)

.....

Prof. Dr. Aldebaro Barreto da Rocha Klautau Jr (CO-ORIENTADOR - UFPA)

.....

Prof. Dr. Ronaldo Freitas Zampolo (MEMBRO - UFPA)

.....

Prof. Dr. Licinius Dmitri Sá de Alcântara (MEMBRO EXTERNO - UFRA)

VISTO:

.....

Prof. Dr. Marcus Vinícius Alves Nunes

COORDENADOR DO PPGEE/ITEC/UFPA

Quero, um dia, poder dizer às pessoas que nada foi em vão...

Que o amor existe, que vale a pena se doar às amizades e às pessoas,

que a vida é bela sim, e que eu sempre dei o melhor de mim...

e que valeu a pena.

(Mário Quintana)

Agradecimentos

Agradeço primeiramente à minha família, em especial a minha mãe Sandra e meu pai Marcos por sempre me incentivarem a estudar e pelo amor e apoio que têm me dado durante a vida. Ao meu irmão Gustavo e irmã Hannah pelo incentivo.

Agradeço ao Francisco pelo carinho, amor, paciência e incentivo durante a elaboração desse trabalho, mas principalmente por estar sempre disposto a me ajudar nas horas que mais precisei.

Agradeço a todos os amigos do Laboratório de Processamento de Sinais pela companhia e por fazerem o ambiente de trabalho um lugar agradável. Em especial aos colegas do projeto Ericsson pela ajuda e por sempre estarem dispostos a tirar minhas dúvidas.

Aos meus amigos de fora da UFPA, em especial aos da Cia Compassos pela compressão e apoio.

Agradeço ao Prof. Aldebaro Klautau, pela co-orientação na elaboração desse trabalho, por acreditar no meu trabalho, pelo apoio e incentivo, mesmo nas horas em que fui teimosa, querendo fazer as coisas do jeito mais complicado.

Um agradecimento todo especial ao meu orientador, Prof. Evaldo Pelaes, por ser muita mais do que um orientador, sempre atensioso e disposto a tirar minha dúvidas. Agradeço pela confiança em mim.

Agradeço a todos aqueles que direta ou indiretamente contribuiram para a conclusão de mais essa fase na minha vida.

Resumo

Este trabalho apresenta a implementação em software da codificação de canal utilizada no padrão ADSL. A teoria da codificação de canal é descrita, bem como a codificação de canal implementada no Software Modem ADSL utilizando o ambiente de desenvolvimento Ptolemy II. A implementação de um modelo de ruído impulsivo também é apresentada. Para garantir que a implementação obedeça o padrão do ADSL, testes utilizando o analisador de sistemas DSL TraceSpan são descritos. O trabalho apresenta ainda um exemplo de aplicação do Software Modem ADSL, caracterizado por um estudo de caso sobre os efeitos do ruído impulsivo na transmissão de vídeo, analisando o impacto de alguns parâmetros da codificação de canal na correção dos erros.

PALAVRAS-CHAVE: Codificação de canal, Software Modem, Ptolemy, ADSL, transmissão de vídeo.

Abstract

This work describes the implementation in software of the channel coding used in the ADSL standard. The theory of channel coding is described, as well as the channel coding implemented in the Software Modem ADSL using the development environment Ptolemy II. The implementation of a impulsive noise model is also presented. To ensure that the implementation meets the ADSL standard, tests using the DSL systems analyzer TraceSpan are described. This work also presents an application example of the Software Modem ADSL, characterized by a study case on the effects of impulsive noise in the video transmission, analyzing the impact of some parameters of channel coding in the correction of errors.

KEYWORDS: Channel coding, Software Modem, Ptolemy, ADSL, video transmission

Lista de Figuras

Lista de Tabelas

Lista de Siglas

DSL	-	*Digital Subscriber Line*
ADSL	-	*Asymmetric Digital Subscriber Line*
VDSL	-	*Very High Speed Digital Subscriber Line*
HDSL	-	*High bit rate Digital Subscriber Line*
SDSL	-	*Symetric Digital Subscriber Lines*
SHDSL	-	*Single Pair High-Speed Digital Subscriber Line*
DSM	-	*Dynamic Spectrum Management*
FEC	-	*Foward Error Correction*
PLC	-	*Power Line Communication*
DSLAM	-	*Digital Subscriber Line Access Multiplexer*
ATU-C	-	*ADSL Transceiver Unit at the Central Office*
ATU-R	-	*ADSL Transceiver Unit at the Remote Terminal*
ANSI	-	*American National Standards Institute*
ITU	-	*International Telecommunication Union*
DMT	-	*Discrete Multitone*
SNR	-	*Signal-to-Noise Ratio*
QAM	-	*Quadrature Amplitude Modulation*
IFFT	-	*Inverse Fast Fourier Transformation*
CP	-	*Cyclic Prefix*
ISI	-	*Intersymbol Interference*
ICI	-	*Intercarrier Interference*
TEQ	-	*Time Domain Equalizer*

FFT - *Fast Fourier Transformation*

FEQ - *Frequency Domain Equalizer*

NEXT - *Near-End Crosstalk*

FEXT - *Far-End Crosstalk*

BT - *British Telecom*

DT - *Deutsche Telekon*

MNLT - *Memoryless Non-Linear Transform*

ARQ - *Automatic Request for Retransmission*

GF - *Galois Field*

CRC - *Cyclic Redundancy Check*

BCH - *Bose-Chaudhuri-Hocquenghem*

RS - *Reed-Solomon*

EOC - *Embedded Operations Channel*

AOC - *ADSL Overhead Control Channel*

LFSR - *Linear Feedback Shift Register*

DE - *Discrete Event*

CT - *Continuos Time*

SDF - *Synchronous Dataflow*

FSM - *Finite State Machine*

CSP - *Communicating Sequential Processes*

PN - *Process Network*

DDF - *Dynamic Dataflow*

IPTV - *Internet Protocol Television*

PSD - *Power Spectral Density*

MOS - *Mean Opinion Score*

PSNR - *Peak Signal-to-Noise Ratio*

MSE - *Mean Square Error*

Lista de Símbolos

$f_i(u)$	-	Função densidade de probabilidade exponencial para as amplitudes do ruído impulsivo
$P(y)$	-	Função de densidade Weibull modificada para as amplitudes do ruído impulsivo
α, b	-	Parâmetros do modelo de amplitudes do ruído impulsivo
$f_l(t)$	-	Função densidade de probabilidade para a duração do ruído impulsivo
B, s_1, t_1, s_2, t_2	-	Parâmetros do modelo de duração do ruído impulsivo
k	-	Número de bits de informação
n	-	Tamanho da palavra-código
$n-k$	-	Bits de paridade ou redundância
p	-	Número de elementos de um campo finito de Galois
m	-	Número de bits
$C(n,k)$	-	Código de bloco linear
$\mathbf{G}$	-	Matriz Geradora $k \times n$
$\mathbf{I_k}$	-	Matriz identidade $k \times k$
$\mathbf{P}$	-	Sub-matriz de paridade $k \times (n-k)$
$\mathbf{H}$	-	Matriz de verificação de paridade $(n-k) \times k$
$\mathbf{I_{n-k}}$	-	Matriz identidade $k \times k$
$\mathbf{P}^T$	-	Transposta da matriz de coeficientes
$\mathbf{H}^T$	-	Transposta da matriz de verificação de paridade
$\mathbf{r}$	-	Vetor da palavra recebida
$\mathbf{u}$	-	Vetor da palavra codificada transmitida
$\mathbf{e}$	-	Vetor de erros
$\mathbf{s}$	-	Vetor Síndrome

$d(\mathbf{u}, \mathbf{v})$	-	Distância de *Hamming* entre dois vetores **u** e **v**
$w(\mathbf{u})$	-	Peso de *Hamming* de um vetor **u**
d_{min}	-	Distância mínima de *Hamming*
t	-	Capacidade de correção de um código
$\mathbf{c}$	-	Vetor palavra-código
i	-	Índice de posições
$c(X)$	-	Representação polinomial do vetor código **c**
$c^{(i)}(X)$	-	Deslocamento cíclico pra direita de i posições de um vetor **c** representado na forma polinomial
$g(x)$	-	Polinômio gerador
$m(x)$	-	Polinômio da mensagem
$b(x)$	-	Polinômio do resto da divisão
$h(x)$	-	Polinômio do verificação de paridade
$h_r(X)$	-	Polinômio recíproco de $h(x)$
$r(X)$	-	Representação polinomial da palavra recebida **r**
$q(x)$	-	Representação polinomial do quociente
$s(x)$	-	Polinômio da síndrome
$e(x)$	-	Polinômio de erros
K	-	Número de bytes ou símbolos de informação
N	-	Tamanho da palavra-código em bytes ou símbolos
$RS(N, K)$	-	Código Reed-Solomon com N igual ao número total de símbolos codificados e K o número de símbolos de informação
β	-	Elemento primiritvo de um Campo de Galois
R	-	Bytes ou símbolos de redundância ou paridade
$GF(p)$	-	Campode Galois com p elementos
$C(X)$	-	Palavra-código de um Código RS na forma polinomial
$M(X)$	-	Polinômio mensagem de um Código RS
$G(X)$	-	Polinômio gerador para um Código RS
$R(X)$	-	Polinômio paridade de um Código RS

$\sigma_{BM}^{(\mu)}(X)$	-	Polinômio Localizador de Erros do algoritmo Berlekamp-Massey
l_μ	-	Grau do polinômio $\sigma_{BM}^{(\mu)}(X)$
d_μ	-	μ-ésima discrepância
$p(X)$	-	Polinômio primitivo de um Código RS
K_F	-	Número de bytes de informação do *fast buffer*
N_F	-	Tamanho da palavra-código em bytes do *fast buffer*
R_F	-	Número de bytes de redundância do *fast buffer*
K_I	-	Número de bytes de informação do *interleaved buffer*
N_I	-	Tamanho da palavra-código em bytes do *interleaved buffer*
R_I	-	Número de bytes de redundância do *interleaved buffer*
S	-	Número de frames ou símbolos DMT que podem ser colocados em uma palavra-código RS
D	-	*Interleaver Depth*

Sumário

Capítulo 1

Introdução

Com a crescente demanda por maiores taxas transmissão de dados, novas tecnologias surgiram ao longo dos últimos anos para possibilitar acesso rápido e barato à Internet, visando principalmente usuários domésticos. Nesse sentido, um conjunto de tecnologias se destaca: o DSL (do inglês *Digital Subscriber Line*), ou linha digital do assinante. Todas as tecnologias desta família utilizam as linhas telefônicas convencionais para transmissão de dados em alta velocidade, sem afetar o serviço de comunicação por voz. A maior vantagem no uso das linhas telefônicas é o aproveitamento de uma infraestrutura já existente em boa parte do mundo e de fácil manutenção, apesar das limitações impostas por um meio de transmissão que foi projetado apenas para uso na comunicação de voz.

Entre as tecnologias DSL, umas das mais populares é a tecnologia ADSL (do inglês *Asymmetric Digital Subscriber Line*). Essa tecnologia oferece taxas de transmissão de dados de até 9 Mb/s da central para o usuário e de até 1 Mb/s do usuário para a central. Daí vem o nome *asymmetric*, pois as taxas de transmissão de dados são diferentes dependendo da direção da transmissão [1].

O uso das linhas telefônicas impõe uma série de limitações na transmissão de dados, sendo as principais as interferências causadas por *crosstalk* e ruído impulsivo. O *crosstalk* surge devido à interferência eletromagnética entre os pares trançados das linhas telefônicas, enquanto o ruído impulsivo é causado por eventos eletromagnéticos temporários que ocorrem na vizinhança das linhas telefônicas [1].

Diversas técnicas são utilizadas para combater essas interferências. No caso específico do *crosstalk*, é utilizado o gerenciamento de espectro. Esse gerenciamento pode ser estático, onde cada linha usa máscaras de transmissão de potência ao longo de frequências pré-definidas, ou dinâmico, onde essas máscaras são definidas de acordo com as condições atuais da rede telefônica. O gerenciamento dinâmico do espectro é chamado de DSM, do inglês *Dynamic*

Spectrum Management [2].

De uma maneira mais geral, a degradação causada pelos diferentes tipos de interferências e ruídos é combatida com o uso de técnicas de codificação de canal, que utilizam códigos corretores de erros para detectar e corrigir erros ocorridos durante a transmissão. Um código corretor de erro é um modo organizado de adicionar dados a cada informação que se queira transmitir ou armazenar com o intuito de detectar e corrigir erros durante a recuperação dessa informação. A grande vantagem no uso da codificação de canal é permitir o uso de menos potência durante a transmissão de dados para uma dada probabilidade de erros, quando comparado à transmissão sem o uso de codificação. Isso é especialmente importante em sistemas DSL, onde o *crosstalk* é diretamente proporcional à potência utilizada para a transmissão.

Além disso, a codificação de canal é fundamental no combate ao ruído impulsivo. No caso de sistemas ADSL, é utilizada a correção direta de erros (FEC - *Foward Error Correction*), que faz uso dos códigos Reed-Solomon como método para detecção e correção dos erros, juntamente com o uso do *interleaver*. A função do *interleaver* é espalhar os erros de modo a tornar possível a correção usando o Reed-Solomon, pois o ruído impulsivo tende a causar erros em rajada.

Este trabalho apresenta a implementação em software da codificação de canal utilizada no padrão ADSL [3], incluindo um estudo de caso sobre os efeitos do ruído impulsivo na transmissão de vídeo. A implementação em software é realizada utilizando o Ptolemy II [4], um *framework open-source* construído em Java que permite a simulação de diversos sistemas. É importante ressaltar que a implementação da codificação de canal do ADSL está inserida dentro de um projeto maior, que busca construir um modem completo em software seguindo o padrão ADSL.

1.1 Motivação

A implementação de um modem seguindo o padrão ADSL em software apresenta diversos benefícios. Um dos mais imediatos é a possibilidade da modificação de parâmetros de transmissão que não são acessíveis em um modem comercial. O ajuste desses parâmetros permite o teste do modem em condições não previstas pelo padrão como, por exemplo, quando algoritmos de gerenciamento dinâmico de espectro são utilizados. O modem em software pode ser expandido e modificado de maneiras que não seriam possíveis no caso do modem em hardware. Outra utilidade do modem ADSL em software é servir como ferramenta de aprendizado e pesquisa, permitindo um maior entendimento do funcionamento do modem,

além de possibilitar o teste de novos algoritmos com facilidade. Além disso, padrões mais recentes como o VDSL2 [5] podem ser implementados usando o modem em software para o padrão ADSL como base.

Um exemplo de implementação em software de um sistemas ADSL2 utilizando o LabVIEW é mostrado em [6].

Como mencionado anteriormente, o foco desse trabalho é a codificação de canal do padrão ADSL. Para demonstrar a funcionalidade da codificação de canal implementada, um estudo de caso sobre os efeitos do ruído impulsivo na transmissão de vídeo é realizado. O ruído impulsivo é uma fonte de degradação importante em sistemas ADSL e a codificação de canal é fundamental para minimizar seus efeitos deletérios. O uso do software modem permite a variação de parâmetros relacionados aos blocos de codificação de canal, possibilitando a análise do efeito do uso de diferentes configurações na qualidade da recepção de vídeos na presença de ruído impulsivo.

1.2 Objetivos

Os objetivos específicos dessa dissertação são:

- Implementar a codificação de canal de um modem ADSL descrito no padrão [3] utilizando o *framework* Ptolemy II;
- Mostrar que os blocos implementados seguem o padrão ADSL de acordo com testes realizados utilizando o analisador de sistemas DSL TraceSpan;
- Como exemplo de aplicação do modem, é mostrado um estudo de caso de transmissão de vídeo na presença de ruído impulsivo. Parâmetros dos blocos de codificação de canal são alterados e seu efeito na qualidade dos vídeos recebidos é descrito.

1.3 Organização do Trabalho

Este trabalho está organizado como descrito a seguir:

- **Capítulo 2:** Este capítulo apresenta uma introdução aos sistemas DSL com ênfase no sistema ADSL. Além disso, o capítulo descreve as interferências que atingem os sistemas ADSL, como *crosstalk* e ruído impulsivo. O modelo de ruído impulsivo adotado e implementado nesse trabalho é também descrito nesse capítulo.

- **Capítulo 3:** Este capítulo apresenta um estudo sobre a teoria da codificação de canal e controle de erros. Os códigos corretores de erros, como os códigos de blocos lineares e os códigos cíclicos são descritos, incluindo a fundamentação teórica dos códigos Reed-Solomon.

- **Capítulo 4:** Este capítulo trata da codificação de canal e controle de erros utilizada no ADSL, mostrando o esquema de um modem ADSL, descrevendo os blocos responsáveis pela codificação de canal.

- **Capítulo 5:** Este capítulo descreve a implementação da codificação de canal do ADSL utilizando o *framework* Ptolemy II, incluindo uma descrição do funcionamento do *framework* e uma descrição da implementação dos blocos de codificação de canal. Além disso, este capítulo descreve os testes realizados com o analisador TraceSpan para garantir que os blocos implementados seguem o padrão ADSL, e a descrição da implementação do modelo de ruído impulsivo adotado.

- **Capítulo 6:** Este capítulo descreve os resultados obtidos utilizando a transmissão de vídeo na presença de ruído impulsivo como exemplo de aplicação do software modem ADSL implementado utilizando Ptolemy, enfatizando os blocos de codificação de canal. Alguns parâmetros dos blocos de codificação de canal são alterados e o efeito do ruído impulsivo na qualidade dos vídeos é analisado.

- **Capítulo 7:** Neste capítulo são apresentadas as conclusões e sugestões para trabalhos futuros.

Capítulo 2

Sistemas DSL

2.1 Introdução aos Sistemas DSL

As linhas telefônicas foram inicialmente projetadas para transmitir sinais de voz utilizando uma faixa de frequência de até 4kHz. As tecnologias DSL surgiram na década de 90, desenvolvidas pela Bellcore com o objetivo de transmitir dados em alta velocidade através das mesmas linhas telefônicas, usando uma larga faixa de frequência antes inutilizada. Para alcançar esse objetivo, são aplicadas sofisticadas técnicas de transmissão digital que compensam os diversos distúrbios comuns às linhas telefônicas [1].

A infra-estrutura de cabeamento telefônico conecta hoje quase todas as residências e escritórios pelo mundo. Com o uso dessa infra-estrutura, as tecnologias DSL acabam sendo economicamente mais viáveis, capazes de satisfazer as necessidades atuais de alta velocidade. São as mais utilizadas no Brasil em comparação com as outras tecnologias, como banda larga móvel, modems a cabo, PLC e etc.

Além de possibilitar o acesso à Internet de alta velocidade e a interconexão de redes locais, as tecnologias DSL apresentam diversas outras aplicações, incluindo aplicações em tempo real, transmissão de vídeo, videoconferência, aprendizado a distância, sistemas interativos e etc.

2.1.1 Tecnologias DSL

As tecnologias DSL podem ser divididas quanto ao tipo de transmissão: em simétricas, nas quais as taxas de transmissão de *upstream* e *downstream* são iguais; assimétrica, nas quais as taxas de transmissão são diferentes (onde tipicamente a taxa de *downstream* é maior); e as

que são simétricas e assimétricas, operando nos dois modos. A Tabela 2.1 mostra as principais tecnologias DSL, assim como algumas características.

Tabela 2.1: Tecnologias xDSL [7]

xDSL	Tipo	Taxa de *Downstream*	Taxa de *Upstream*	Alcance
HDSL	Simétrico	1.544 Mb/s	1.544 Mb/s	3.6 km
SDSL	Simétrico	2.048 Mb/s	2.048 Mb/s	3.0 km
ADSL	Assimétrico	8.0 Mb/s	1.0 Mb/s	3.6 km
		1.5 Mb/s (*Lite*)	512 kb/s (*Lite*)	5.4 km
SHDSL	Simétrico	2.3 Mb/s	2.3 Mb/s	3.3 km
VDSL	Assimétrico e	52 Mb/s (assimét.)	6 Mb/s (assimét.)	1.5 km
	Simétrico	26 Mb/s (simét.)	26 Mb/s (simét.)	
ADSL2	Assimétrico	12 Mb/s	1.0 Mb/s	5.6 km
ADSL2+	Assimétrico	24 Mb/s	1.0 Mb/s	6.7 km
VDSL2	Assimétrico ou	100 Mb/s	100 Mb/s	3.6 km
	Simétrico			

A tecnologia ADSL é a mais popular das tecnologias mencionadas na Tabela 2.1 e foi utilizada nesse trabalho por ser considerada a mais simples para a construção de um modem em software.

2.2 ADSL

A tecnologia ADSL é uma tecnologia que permite a transferência de dados em alta velocidade utilizando as linhas telefônicas já existentes como meio de transmissão, sem interferir no serviço de voz. O ADSL foi originalmente desenvolvido para aplicações de vídeo sob demanda, no entanto, nos meados dos anos 90, a motivação da tecnologia mudou para o acesso a Internet banda larga, por razões comerciais [8].

No ADSL, a transmissão de dados é assimétrica, ou seja, a taxa de transferência de dados é maior na direção *downstream* (em direção ao assinante), podendo chegar a até 9 Mb/s, do que na direção *upstream* (transmissão do assinante para a rede), podendo chegar a até 1 Mb/s [1]. O ADSL divide a faixa de frequência não utilizada pelos serviços de telefonia, considerando uma faixa de frequência maior para o *downstream*, no pressuposto de que os

usuários da Internet recebem muito mais dados do que enviam. Além disso, o ADSL garante a transmissão de voz e de dados em alta velocidade simultaneamente pela linha, ou seja, é possível navegar pela internet e receber ligações ao mesmo tempo, através de um *splitter*, que separa voz e dados.

No ADSL, as faixas de frequência são divididas para voz, transmissão *upstream* e *downstream* como na Figura 2.1.

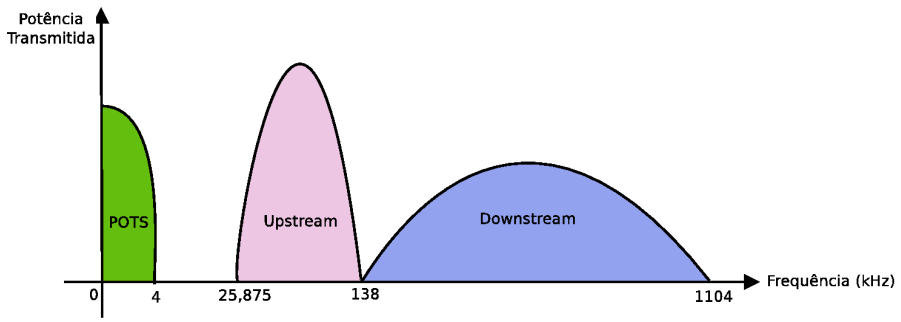


Figura 2.1: Faixas de frequência do ADSL.

A Figura 2.2 mostra o modelo de uma arquitetura ADSL que conecta o usuário a uma rede central. O enlace consiste de, do lado do usuário, um modem do usuário (ATU-R), um *splitter*[1] para dividir o sinal que chega do canal telefônico, e um telefone. No lado da central, o enlace consiste de um *Digital Subscriber Line Access Multiplexer* (DSLAM) que funciona como o transceptor da operadora (ATU-C) e é responsável por se comunicar e gerenciar vários usuários ao mesmo tempo, e um *splitter* para também dividir o sinal entre o DSLAM e a central telefônica.

A tecnologia ADSL tem sido padronizada por organizações internacionais, como por exemplo, a *American National Standards Institute* (ANSI), com o padrão ANSI T1.413 [9], e o *International Telecommunication Union* (ITU), com o padrão ITU-T G.992.1 [3].

2.2.1 Modulação

A técnica de modulação utilizada no ADSL é chamada de modulação discreta por tom (do inglês DMT - *Discrete Multitone*). O DMT particiona o espectro de transmissão em vários subcanais (ou tons) paralelos e independentes [10]. Cada subcanal é alocado com um

[1]Os *splitters* funcionam como filtros passa-baixa separando os sinais de voz e dados. No entanto, a maioria das instalações de ADSL atuais utilizam dispositivos denominados "microfiltros" que são filtros de linha, podendo ser mais facilmente instalados na residência do usuário, sem perda de eficiência [2].

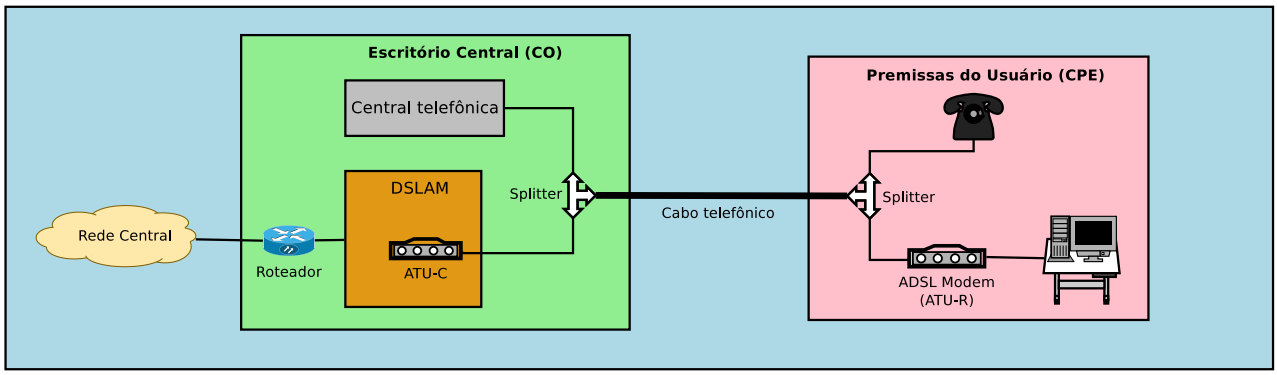


Figura 2.2: Arquitetura de um enlace ADSL.

determinado número de bits especificado de acordo com a razão sinal ruído (SNR), processo chamado de *bitloading*. Ou seja, quanto maior a SNR do subcanal, maior é a quantidade de bits alocados. E para subcanais com menor SNR, menor será a quantidade de bits, como mostrado na Figura 2.3, com o espaçamento entre os subcanais igual a 4,3125 kHz.

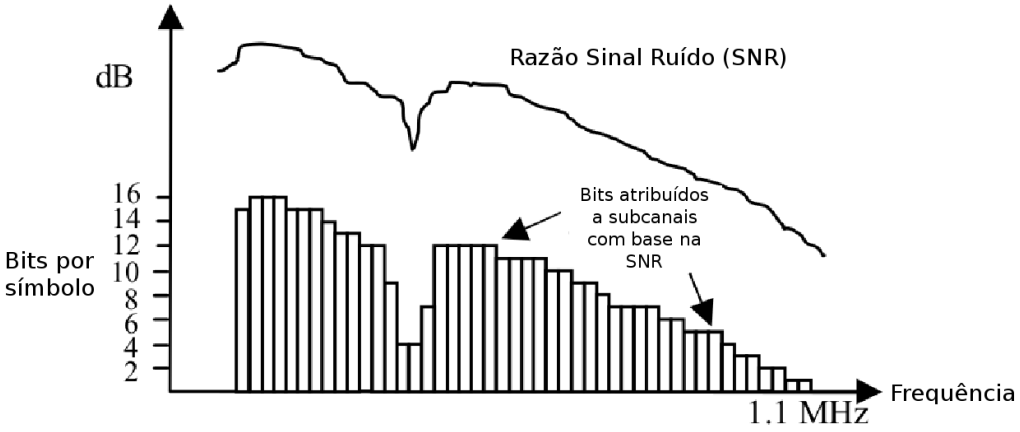


Figura 2.3: Exemplo de modulação DMT, no qual o espectro de frequência é dividido em vários subcanais. E a alocação de bits é baseada na SNR [11].

O ADSL utiliza 256 subcanais na direção *downstream* e 32 subcanais no *upstream*, sendo que cada subcanal suporta um máximo de 15 bits [1].

A modulação DMT em sistemas ADSL utiliza o esquema de modulação QAM (*Quadrature Amplitude Modulation*) [12] em seus subcanais. O esquema QAM é responsável por mapear as sequências de bits em símbolos de uma constelação QAM em cada subcanal. No transmissor de sistemas DMT, os algoritmos de *bitloading* atribuem os bits para cada

subcanal, cada grupo de bits é mapeado em símbolos de uma constelação QAM, esses símbolos são representados por números complexos que através de uma IFFT (*Inverse Fast Fourier Transformation*) são mapeados para um sinal no domínio do tempo. Para que as amostras sejam números reais e transmitidas pelo par trançado, a simetria *hermitiana* é aplicada nos números complexos. Essas amostras no domínio do tempo são chamadas de *símbolo DMT*. Na saída da IFFT é inserido um prefixo cíclico (CP) que cria um intervalo de guarda entre os símbolos pra combater a interferência inter-simbólica (ISI, *Intersymbol Interference*). Além disso, o prefixo cíclico protege contra a interferência entre portadoras (ICI, *Intercarrier Interference*) [8].

No receptor, primeiramente é utilizado um equalizador no domínio do tempo (TEQ, *Time Domain Equalizer*) para garantir que a resposta ao impulso do canal seja menor que o prefixo cíclico. Após a equalização, o prefixo cíclico é descartado e as amostras passam pela FFT (*Fast Fourier Transformation*). Na saída da FFT, um equalizador no domínio da frequência (FEQ, *Frequency Domain Equalizer*) é aplicado com o objetivo de compensar as distorções de amplitude e fase causadas pelo canal. Após o FEQ, os bits passam pelo decodificador QAM sendo então recuperados [8].

2.3 Canal de Comunicação

Um sistema de transmissão de comunicação digital pode ser representado pelo diagrama de blocos da Fig. 2.4.

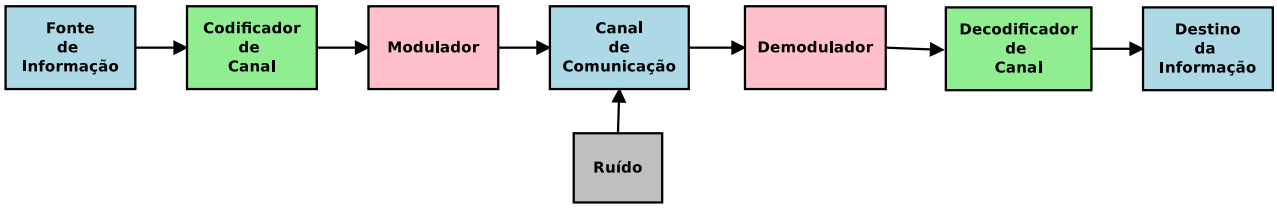


Figura 2.4: Diagrama de blocos de um sistema de transmissão de comunicação digital.

A fonte de informação gera informação na forma de uma sequência de dígitos binários (bits). O codificador no transmissor transforma essa sequência em uma sequência codificada, adicionando bits de redundância, chamada de palavra-código. Os símbolos são mapeados pelo modulador para formas de onda. Essas formas de onda são, então, transmitidas pelo canal. O canal pode sofrer interferências ou ruídos que corrompem a mensagem transmitida. Esses ruídos podem ser causados por diversos fatores. Em uma linha telefônica, por exemplo, a

interferência pode ser por *crosstalk*, ruídos impulsivos, etc. O demodulador é responsável por receber a forma de onda e reproduzir a sequência binária. E o decodificar é responsável por decodificar a palavra-código, usando a redundância para detectar e corrigir os erros.

2.4 Interferências

Os sistemas DSL podem ser afetados por diferentes tipos de interferências que podem comprometer a transmissão. Essas interferências são fatores limitantes na transmissão de dados e podem ser classificadas em limitante de capacidade e limitante de qualidade [13].

As interferências que limitam a capacidade, como por exemplo, o *crosstalk*, são estacionários, variando pouco no tempo, podem ser facilmente previstos e por isso podem ser minimizados ou eliminados com maior facilidade. Já as interferências limitante de qualidade (em taxa de erro de bits) são difíceis de prever, por serem intermitentes e não-estacionárias, e por isso mais difícil de minimizar o seu impacto, como, por exemplo, o ruído impulsivo.

2.4.1 Crosstalk

O *crosstalk* em sistemas DSL ocorre devido ao acoplamento magnético entre os fios em um mesmo cabo de par trançado e é considerado o pior tipo de interferência em um par trançado, podendo reduzir substancialmente a performance de um sistema DSL [1]. O *crosstalk* pode ser classificado em dois tipos: o *near-end crosstalk* (NEXT) e o *far-end crosstalk* (FEXT). O NEXT é a interferência proveniente do usuário localizado no mesmo extremo do cabo e o FEXT é a interferência proveniente do usuário no extremo oposto do cabo, como mostra a Figura 2.5.

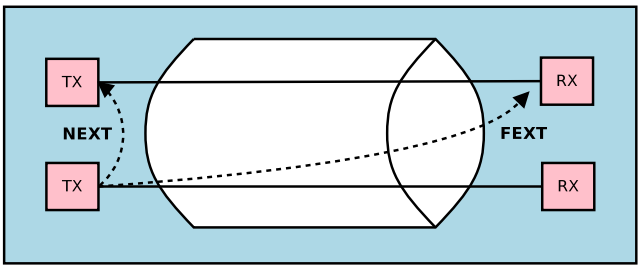


Figura 2.5: Tipos de *crosstalk*: NEXT e FEXT.

A intensidade do NEXT é considerada pior que do FEXT, pois o FEXT sofre atenuação antes de atingir o usuário no outro extremo do cabo.

2.4.2 Ruído Impulsivo

O ruído impulsivo (*Impulsive Noise*) é uma interferência eletromagnética não-estacionária e estocástica que consiste de ocorrências aleatórias de picos de energia com amplitudes e durações variadas [14]. As causas do ruído impulsivo podem ser diversas, incluindo descargas elétricas, equipamentos com motores (eletrodomésticos e eletrônicos), cercas eletrificadas, sinal sonoro do telefone, liga/desliga de lâmpadas fluorescentes, etc. A Figura 2.6 mostra alguns exemplos de ruídos impulsivos que fazem parte de uma campanha de medições.

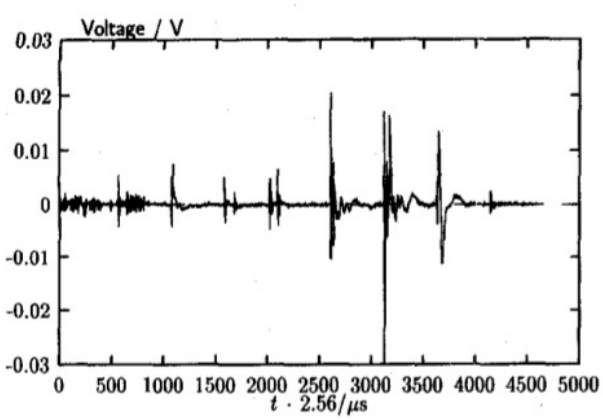


Figura 2.6: Alguns exemplos de ruídos impulsivos medidos em uma cidade alemã [15].

Os erros causados por ruído impulsivo ocorrem na forma de erros em rajada (*bursts*) corrompendo os dados que estão sendo transmitidos pelo cabo telefônico. Em aplicações envolvendo transmissão de vídeo, por exemplo, esses erros são observados na distorção das imagens. Os erros provenientes do ruído impulsivo podem ser combatidos com o uso de técnicas de codificação (FEC), juntamente com o *interleaver*.

O ruído impulsivo é considerado um grande problema devido a irregularidade de ocorrências e intensidade, sendo difícil de ser modelado e previsto. No entanto, alguns modelos foram elaborados a partir das estatísticas do ruído impulsivo, baseados em observações e medidas e que são úteis para testes ou para avaliações do seu impacto. Nesse trabalho será usado um modelo para testar a implementação de um Software modem ADSL que será descrito na seção seguinte.

2.4.3 Modelo de Ruído Impulsivo Adotado

O modelo de ruído impulsivo adotado nesse trabalho segue a descrição de um modelo recente [16, 17] baseado em medidas realizadas pela British Telecom (BT), Deutsche Telekon (DT) e Universidade de Edinburg para simular ruídos impulsivos realistas.

O ruído impulsivo, no domínio do tempo, é caracterizado pela amplitude e duração do impulso e pelo intervalo entre impulsos chamado de *inter-arrival times*. O modelo de ruído impulsivo aqui adotado modela estaticamente essas características.

De acordo com [16], o processo de geração de uma *stream* de impulsos simulados depende de quatro geradores de números aleatórios:

- gerador de amplitudes do ruído impulsivo;
- gerador de duração de um ruído impulsivo;
- gerador de intervalo entre os impulsos (*inter-arrival times*);
- gerador de espectro do ruído impulsivo.

Cada gerador é baseado nas características de modelagem do ruído impulsivo descritas a seguir.

2.4.3.1 Amplitudes do ruído impulsivo

Inicialmente, para modelar as estatísticas da amplitude do ruído impulsivo, um modelo foi proposto em [18], usando a seguinte distribuição exponencial:

$$f_i(u) = \frac{1}{240u_0} e^{-|u/u_0|^{1/5}}. \tag{2.1}$$

No entanto, um outro modelo foi proposto em [17], por ser mais tratável matematicamente, usando uma função de densidade Weibull modificada para torná-la simétrica, como uma possível alternativa para modelar as amplitudes:

$$P(y) = \frac{1}{2}\alpha b|y|^{\alpha-1} e^{-b|y|^{\alpha}}, \tag{2.2}$$

onde α e b são parâmetros. O modelo citado em [17] será usado nesse trabalho e uma comparação entre os dois modelos pode ser encontrada também em [17].

A Tabela 2.2 lista exemplos de valores para os parâmetros α e b, a partir de medições da BT e DT.

Tabela 2.2: Parâmetros do modelo para a função de densidade de Weibull a partir das medições da BT e DT [17].

	$\boldsymbol{\alpha}$	**b**
BT(CP)	0,263	4,77
DT(CP)	0,486	44,40
DT(CO)	0,216	12,47

2.4.3.2 Duração do ruído impulsivo

A duração do ruído impulsivo é modelada através da soma de duas componentes log-normal, como proposto em [18]:

$$f_l(t) = B\frac{1}{\sqrt{2\pi}s_1 t}e^{-(1/s_1^2)}\ln^2(t/t_1) + (1-B)\frac{1}{\sqrt{2\pi}s_2 t}e^{-(1/s_2^2)}\ln^2(t/t_2), \tag{2.3}$$

onde t é a duração do impulso e B, s_1, t_1, s_2 e t_2 são parâmetros.

A Tabela 2.3 lista os parâmetros dessa distribuição.

Tabela 2.3: Parâmetros do modelo para a função de distribuição da duração do impulso a partir das medições da BT e DT [17].

	B	$\mathbf{s_1}$	$\mathbf{t_1}$	$\mathbf{s_2}$	$\mathbf{t_2}$
BT(CP)	0,45	1,25	1,3μs	21,5	129μs
DT(CP)	1	1,15	18μs	-	-
DT(CO)	0,25	0,75	8μs	1,0	125μs

2.4.3.3 Intervalo entre impulsos (*Inter-arrival times*)

O intervalo entre impulsos (*inter-arrival times*), ou seja, o tempo entre o fim de um impulso e o início do próximo exibe um comportamento estaticamente complexo. A probabilidade de pequenos intervalos entre os impulsos é muito alta, gerando uma cauda de distribuição longa e irregular [14].

Além disso, os erros causados por ruído impulsivo ocorrem em rajadas (*burst*). O comportamento dos intervalos entre ruídos apresentam um grande grau de *clustering*, ou seja,

os impulsos tendem a formar grupos [17, 19]. Por conta desse comportamento, o intervalo entre ruídos pode ser modelado através de um processo de Markov com N estados [19].

Os intervalos entre ruídos são gerados supondo que cada estado representa um intervalo de possíveis *inter-arrival times*, por exemplo, intervalos entre ruídos de 0 a 1 s, e dentro de cada estado existe uma distribuição de probabilidade que gera os possíveis intervalos entre ruídos aleatoriamente, modelados por Poisson ou Pareto. A matriz de transição de probabilidade, então, especifica a probabilidade de transição de um estado para o outro e consequentemente o nível de *clustering* [17].

O número de estados precisa ser definido, assim como a faixa de intervalo entre impulsos para cada estado. Um processo de Markov com 4 estados foi proposto em [19], no qual as faixas eram definidas como (estado 1: 0 a 1 s; estado 2: 1 a 10 s; estado 3: 10 a 100 s e estado 4: 100 s a ∞). Esse modelo fornece uma boa precisão para modelar o intervalo entre ruídos. No entanto, foi mostrado em [17] que um modelo com 2 estados é suficiente para modelar o intervalo entre ruídos com um razoável grau de precisão e grau mínimo de complexidade. Esse modelo é caracterizado para intervalos entre ruídos pequenos, no máximo até 1 s, pois foi mostrado em [19] que a maioria dos impulsos ocorrem nesse intervalo de tempo.

Um modelo com 2 estados foi proposto em [16] para propósitos de simulação. Esse modelo define 2 estados, um com intervalos entre ruídos menores que 1 ms e o outro com intervalos maiores que 1 ms. Cada estado é modelado com uma distribuição de probabilidade. O estado 1, com intervalos menores que 1 ms, é modelado por Poisson e o estado 2 com intervalos maiores que 1 ms por Pareto. A Tabela 2.4 mostra as funções de distribuição de probabilidade usadas por cada estado assim como os parâmetros das distribuições.

Tabela 2.4: Funções de distribuição de probabilidade e parâmetros para cada estado [16].

estado	distribuição	função de distribuição	parâmetros	intervalo
1	Poisson	$\lambda e^{(-\lambda t)}$	$\lambda = 0,16\ \mathsf{ms}^{-1}$	$t < 1\ \mathsf{ms}$
2	Pareto	$\alpha(\frac{t}{1\ \mathsf{ms}})^{-(\alpha-1)}$	$\alpha = 1,5$	$t \geq 1\ \mathsf{ms}$

A matriz de transição de probabilidade com a probabilidade de transição de um estado para o outro é definida por [16]:

$$P = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{bmatrix} = \begin{bmatrix} 0,8 & 0,2 \\ 0,4 & 0,6 \end{bmatrix}. \tag{2.4}$$

Por exemplo, se o estado atual é o estado 1, então a probabilidade de ir para o estado

2 é 0,2.

Uma modelagem das características espectrais do impulso foi proposta em [17]. Usando uma função de auto-correlação, com uma soma de cossenos com decaimento exponencial e um novo método proposto por Tough e Ward [20] que utiliza uma MNLT (*Memoryless Non-Linear Transform*), e gera impulsos com propriedades tanto do domínio do tempo quanto do domínio da frequência. No entanto, esse modelo possui a desvantagem de ser bastante complexo, principalmente no que se refere ao cálculo da MNLT, fazendo com que alguns autores optassem por modelos mais simples, sem considerar as características espectrais do ruído. Um modelo proposto por [14] denominado *Bernoulli-Weibull* baseado na modificação do modelo *Bernoulli-Gaussian*, que considera o ruído como uma soma de uma componente *Bernoulli-Weibull* e uma componente de ruído gaussiano de fundo surge também como uma opção para simulação do ruído impulsivo. No entanto, optamos aqui em utilizar apenas as características no domínio do tempo do ruído sem utilizar as características espectrais, devido à complexidade. Uma descrição mais detalhada de cada uma dessas características do ruído impulsivo pode também ser encontrada em [14].

Capítulo 3

Codificação de Canal

3.1 Introdução à Codificação de Canal

Os canais de comunicação estão sujeitos a diversos ruídos e interferências que introduzem erros durante a transmissão, corrompendo a informação transmitida. As técnicas de detecção de erro possibilitam detectar esses erros, enquanto que as técnicas de correção de erros permitem que esses erros sejam corrigidos e a informação recuperada. Portanto, torna-se necessário o controle de erros sobre os dados transmitidos.

Este capítulo apresenta um estudo sobre a teoria da codificação e controle de erros.

3.2 Teoria dos Códigos

A necessidade de transmissão de dados de um lugar para outro de maneira eficiente, sem sacrificar a taxa de transmissão de dados e garantindo qualidade e confiabilidade na recepção, é um dos problemas enfrentados pelos projetistas de sistemas de comunicação. Quando o canal de comunicação apresenta interferências e ruídos, erros são gerados nos dados transmitidos e a preocupação com o controle de erros se faz necessária para a garantia da qualidade na reprodução dos dados seja preservada. A teoria dos códigos surgiu para resolver esses problemas e desenvolver métodos que permitam detectar e corrigir esses erros.

A Teoria da codificação teve início em 1948 com Shannon [21] e a publicação de um artigo que também deu inicio a Teoria da Informação. Juntamente com a introdução dos códigos de Hamming [22] também na década de 50, e um pouco depois com a invenção dos códigos de Golay [23], a teoria dos códigos se consolidou, e hoje os códigos além de oferecerem

confiabilidade aos canais de comunicação e computadores, são usados em diversas aplicações, como por exemplo, nas comunicações móveis, nos aparelhos de armazenamento de dados (CDs, DVDs), no processamento digital de imagens, comunicação via satélite, Internet, rádio e etc.

Existem dois tipos de técnicas de controle de erros. A correção direta de erros (FEC) baseada nos códigos corretores de erros, que utilizam a redundância na palavra-código transmitida tanto para detecção como para correção de erros durante uma transmissão. E a solicitação de repetição automática (ARQ - *Automatic Request for Retransmission*) que utiliza a redundância na palavra-código apenas para detecção de erros. Depois de detectado o erro, o receptor solicita a retransmissão da palavra-código que foi corrompida [24].

3.2.1 Códigos Corretores de Erros

Os códigos corretores de erros podem ser classificados em dois tipos: códigos de blocos e códigos convolucionais. A diferença está na utilização ou não de codificadores com memória. Os códigos convolucionais, por exemplo, possuem memória. Já os códigos de blocos processam a informação bloco a bloco, tratando cada bloco de informação de maneira independente.

3.2.2 Códigos de Blocos Lineares

Para se entender como funciona a codificação de uma maneira geral, é necessário saber como a informação é transmitida. Uma mensagem ou informação geralmente é transmitida na forma de dígitos binários ou bits, ou seja, 0s e 1s. Chama-se de *palavra* uma sequência de bits. Um código binário, por exemplo, é um conjunto de palavras. E mais especificamente, as *palavras-código* são as palavras pertencentes a esse códigos binários.

Um código é dito linear *se duas palavras-código quaisquer do código puderem ser somadas em aritmética módulo 2 para produzir uma terceira palavra-código* [25].

Em um codificador de bloco linear, a informação é quebrada em uma sequência de blocos de k bits de informação cada. Os códigos de blocos são na verdade, um conjunto de 2^k palavras códigos distintas, e são especificados como códigos *(n,k)*, onde k é o número de bits de informação e n, o tamanho da palavra código. Para gerar um código de bloco, o codificador de canal adiciona de acordo com uma regra de codificação predefinida, *n - k bits de paridade ou redundância* [25]. A adição dessa redundância é que garante que no decodificar a informação possa ser recuperada. Os códigos em que os bits de informação são transmitidos de forma inalterada, e os bits de paridade aparecem no final da palavra codificada, são chamados de *sistemáticos*, como mostrado na Figura. 3.1.

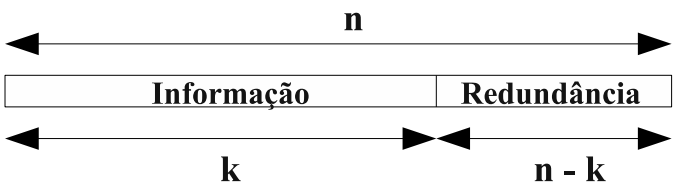


Figura 3.1: Palavra-código em formato Sistemático.

De acordo com Shu Lin [26], um código de blocos linear *C(n,k)* é um subespaço k-dimensional de um espaço vetorial definido em um campo binário ou Campo de Galois. Um Campo de Galois (*Galois Field* - GF) é um campo finito de p elementos, designado por *GF(p)*, em que as operações de adição e multiplicação podem ser executadas. Na prática, para o caso binário, o campo de Galois é definido como $GF(2^m)$, sendo m o número de bits.

3.2.2.1 Matriz Geradora

Cada código de bloco linear tem uma matriz geradora G de onde se obtém as várias palavras do código. A matriz geradora $\mathbf{G}$ de um código *C(n,k)* é uma matriz de dimensão $k \times n$, caracterizada por um conjunto de k linhas linearmente independentes entre si, que formam uma base para C:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & g_{02} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & g_{12} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdots & g_{k-1,n-1} \end{bmatrix}, \tag{3.1}$$

onde $\mathbf{g}_i = (g_{i0}, g_{i1}, ..., g_{i,n-1})$, para $0 \leq i < k$. Considerando $\mathbf{u} = (u_0, u_1, ..., u_{k-1})$, uma mensagem a ser codificada, a palavra-código resultante $\mathbf{v}$ pode ser expressa da seguinte forma [26]:

$$\mathbf{v} = \mathbf{u} \cdot \mathbf{G} = (u_0, u_1, ..., u_{k-1}) \cdot \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + ... + u_{k-1}\mathbf{g}_{k-1}. \tag{3.2}$$

Para um código linear sistemático (n, k), no qual a informação é inalterada e a redundância adicionada no inicio ou no final da mensagem, a matriz geradora $\mathbf{G}$ é composta

de duas sub-matrizes, uma matriz identidade $k \times k$, indicada por $\mathbf{I}_k$ e uma sub-matriz de paridade $k \times (n-k)$, indicada por $\mathbf{P}$, tal que [27],

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_k & \mathbf{P} \end{bmatrix}, \tag{3.3}$$

onde

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & & p_{00} & p_{01} & \dots & p_{0,n-k-1} \\ 0 & 1 & 0 & \dots & 0 & & p_{10} & p_{11} & \dots & p_{1,n-k-1} \\ 0 & 0 & 1 & \dots & 0 & & p_{20} & p_{21} & \dots & p_{2,n-k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \dots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 & & p_{k-1,0} & p_{k-1,1} & \dots & p_{k-1,n-k-1} \end{bmatrix}. \tag{3.4}$$

Nessa notação usa-se a redundância adicionada no final, portanto com a matriz identidade na frente, $\mathbf{G} = [\mathbf{I}_k\mathbf{P}]$, para que a redundância apareça no inicio da mensagem é necessário apenas inverter a posição das sub-matrizes, $\mathbf{G} = [\mathbf{P}\mathbf{I}_k]$, seguindo a notação adotada em [26], por exemplo.

Considerando um código linear (7,4), com a seguinte matriz geradora,

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{3.5}$$

E a informação a ser codificada como sendo $\mathbf{u} = (1101)$, a palavra-código resultante então seria,

$$\mathbf{v} = 1 \cdot \mathbf{g}_0 + 1 \cdot \mathbf{g}_1 + 0 \cdot \mathbf{g}_2 + 1 \cdot \mathbf{g}_3 = (1000110) + (0100011) + (0001101) = (1101000). \tag{3.6}$$

O código linear (7,4), com a redundância adicionado no final da palavra-código é mostrado na Tabela 3.1.

3.2.2.2 Matriz de Verificação de Paridade

Para cada matriz $\mathbf{G}$ de $k \times n$, existe uma matriz $(n-k) \times k$ denominada de matriz $\mathbf{H}$ ou *matriz de verificação de paridade*. As $n-k$ linhas de $\mathbf{H}$ são linearmente independentes, e cada vetor do espaço das linhas da matriz $\mathbf{G}$ é ortogonal as linhas da matriz $\mathbf{H}$ e vice-versa, ou seja, $\mathbf{v}$ só é uma palavra-código de um código gerado por $\mathbf{G}$, se e somente se, $\mathbf{v} \cdot \mathbf{H}^T = 0$ [26].

Tabela 3.1: Código de bloco linear com $k = 4$ e $n = 7$.

Informação	**Palavras-código**
(0 0 0 0)	(0 0 0 0 0 0 0)
(1 0 0 0)	(1 0 0 0 1 1 0)
(0 1 0 0)	(0 1 0 0 0 1 1)
(1 1 0 0)	(1 1 0 0 1 0 1)
(0 0 1 0)	(0 0 1 0 1 1 1)
(1 0 1 0)	(1 0 1 0 0 0 1)
(0 1 1 0)	(0 1 1 0 1 0 0)
(1 1 1 0)	(1 1 1 0 0 1 0)
(0 0 0 1)	(0 0 0 1 1 0 1)
(1 0 0 1)	(1 0 0 1 0 1 1)
(0 1 0 1)	(0 1 0 1 1 1 0)
(1 1 0 1)	(1 1 0 1 0 0 0)
(0 0 1 1)	(0 0 1 1 0 1 0)
(1 0 1 1)	(1 0 1 1 1 0 0)
(0 1 1 1)	(0 1 1 1 0 0 1)
(1 1 1 1)	(1 1 1 1 1 1 1)

Se a matriz $\mathbf{G}$ for sistemática, então a matriz $\mathbf{H}$ é composta de duas sub-matrizes, uma matriz identidade $(n-k)\times(n-k)$, indicada por $\mathbf{I}_{n-k}$, e a transposta da matriz de coeficientes $\mathbf{P}$, indicada por $\mathbf{P}^T$, tal que

$$\mathbf{H} = [\mathbf{P}^T \mathbf{I}_{n-k}] = \begin{bmatrix} p_{00} & p_{10} & \cdots & p_{k-1,0} & 1 & 0 & 0 & \cdots & 0 \\ p_{01} & p_{11} & \cdots & p_{k-1,1} & 0 & 1 & 0 & \cdots & 0 \\ p_{02} & p_{12} & \cdots & p_{k-1,2} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{0,n-k-1} & p_{1,n-k-1} & \cdots & p_{k-1,n-k-1} & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{3.7}$$

A matriz $\mathbf{H}$ é usada para verificar se a palavra-código recebida possui erros ou não, sendo usada na decodificação no receptor. Se considerando uma palavra recebida $\mathbf{r}$ e $\mathbf{r}\cdot\mathbf{H}^T$ for diferente de zero, então $\mathbf{r}$ não é palavra-código e portanto erros ocorreram na transmissão.

A essa verificação dá-se o nome de *síndrome*, a qual será mostrada a seguir.

3.2.2.3 Síndromes

A informação quando atravessa um canal ruidoso pode chegar no receptor contendo alguns possíveis erros. Esses erros não são conhecidos pelo receptor e nem facilmente identificados. A *síndrome* é então usada para definir se a palavra recebida possui erros ou não. Considerando $\mathbf{u}$ a palavra transmitida e $\mathbf{e}$ o vetor de erros, então $\mathbf{r}$, a palavra recebida, é a soma vetorial de $\mathbf{u}$ e $\mathbf{e}$,

$$\mathbf{r} = \mathbf{u} + \mathbf{e}. \tag{3.8}$$

Para recuperar a palavra transmitida $\mathbf{u}$, o decodificador calcula então a *síndrome* da palavra recebida. A *síndrome* é um vetor de $(n-k)$ componentes, definido por,

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H^T}. \tag{3.9}$$

Se $\mathbf{s} = 0$, então $\mathbf{r}$ é uma palavra-código e portanto assume-se que não ocorreram erros na transmissão, no entanto, se $\mathbf{s} \neq 0$, então $\mathbf{r}$ não é palavra-código e erros ocorreram, como mostrado no exemplo a seguir:

Considerando a matriz geradora, apresentada em 3.5 de um código linear (7,4), e sua correspondente matriz $\mathbf{H}$ como sendo,

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{3.10}$$

Supondo que a palavra recebida foi $\mathbf{r} = (1111111)$. A síndrome então seria,

$$\mathbf{s} = (1111111) \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = (000).$$

Nesse caso, como a síndrome é nula, a palavra recebida é palavra-código, como pode ser visto na Tabela 3.1 e assume-se que não ocorreram erros na transmissão. No entanto se a palavra

recebida foi $\mathbf{r} = (1111011)$, a síndrome,

$$\mathbf{s} = (1111011) \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = (100),$$

não é nula. Portanto conclui-se que erros foram introduzidos na transmissão.

3.2.2.4 Capacidade de Correção de Erros de um Código Linear

Antes de definir a capacidade de correção de erros de um código linear, é necessário definir algumas de suas características básicas.

- A distância de *Hamming* entre dois vetores, $d(\mathbf{u}, \mathbf{v})$, é definida como sendo o número de posições em que eles diferem, ou seja, considerando dois vetores $\mathbf{u} = (10101)$ e $\mathbf{v} = (01101)$, a distância de Hamming entre eles é $d(\mathbf{u}, \mathbf{v}) = 2$.
- O peso de *Hamming* de um vetor, $w(\mathbf{u})$, é definido como o número de elementos diferentes de zero do vetor, ou seja, considerando o vetor $\mathbf{u} = (10101)$, o seu peso de Hamming é $w(\mathbf{u}) = 3$.
- A distância mínima de *Hamming* de um código linear, d_{min}, é a menor distância de Hamming encontrada entre todos os pares de palavras de um código.

Se uma palavra recebida possuir o mesmo número de erros ou mais erros que a distância mínima, ou seja, se comparando a palavra recebida com a palavra transmitida, a quantidade de posições em que elas diferem for maior ou igual a distância mínima é possível que coincidentemente essa palavra corresponda a uma outra palavra válida do código, fazendo com que os erros não sejam detectados. Portanto, a detecção de erros é sempre possível quando o número de erros de transmissão numa palavra-código é inferior a distância mínima, ou seja, $d_{min} - 1$, para garantir que a palavra recebida não resulte em uma palavra-código válida.

Se um código corrige até t erros por palavra, então a capacidade de correção de um código é definida por:

$$t = \frac{d_{min} - 1}{2}, \tag{3.11}$$

como pode ser demonstrado em [24, 26].

3.2.2.5 Decodificação

A correção dos erros pode ser feita através de alguns métodos de decodificação. Um dos métodos mais simples de decodificação é o chamado "algoritmo do vizinho mais próximo", no qual o objetivo é achar a palavra recebida que mais se aproxime da transmitida. Considere **u** a palavra codificada transmitida, e **r** a palavra recebida, nesse algoritmo calcula-se a probabilidade de **r** ser **u**, sendo que a palavra **r** será decodificada para aquela com a maior probabilidade.

Esse tipo de decodificação pode ser feita para códigos pequenos, porém para código maiores, comparar todas as palavras-código do código para achar a que mais se aproxime torna-se inviável.

Uma outra forma de decodificação é a chamada "decodificação por síndromes" [26]. Na decodificação por síndromes utiliza-se um *Arranjo Padrão* dividindo os 2^n possíveis vetores recebidos em 2^k sub-conjuntos disjuntos. Para formar o Arranjo Padrão, primeiro coloca-se as 2^k palavras-código do código **C** na primeira linha da matriz, com a palavra código de somente zeros c_1 como o primeiro elemento a esquerda. A segunda linha é formada pela adição de um padrão de erro $\mathbf{e}_2$ a cada palavra-código da primeira linha e assim sucessivamente até que todos os padrões de erros sejam usados. Formando assim o arranjo padrão com 2^{n-k} linhas e 2^k colunas, como mostrado na matriz 3.12:

$$\begin{array}{ccccccc} c_1 = 0 & c_2 & c_3 & \dots & c_i & \dots & c_{2^k} \\ e_2 & c_2 + e_2 & c_3 + e_2 & \dots & c_i + e_2 & \dots & c_{2^k} + e_2 \\ e_3 & c_2 + e_3 & c_3 + e_3 & \dots & c_i + e_3 & \dots & c_{2^k} + e_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ e_j & c_2 + e_j & c_3 + e_j & \dots & c_i + e_j & \dots & c_{2^k} + e_j \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ e_{2^{n-k}} & c_2 + e_{2^{n-k}} & c_3 + e_{2^{n-k}} & \dots & c_i + e_{2^{n-k}} & \dots & c_{2^k} + e_{2^{n-k}} \end{array} \tag{3.12}$$

As 2^{n-k} linhas são chamadas de *cosets* e o primeiro elemento de cada *coset* é chamado de *líderes de coset*. A síndrome de qualquer vetor do *coset* é igual a síndrome do *líder do coset* e para cada padrão de erro $\mathbf{e_i}$ existe uma síndrome diferente correspondente. Portanto, para realizar a decodificação executam-se os seguintes passos:

- Calcula-se a síndrome da palavra recebida **r**, $\mathbf{s} = \mathbf{r} \cdot \mathbf{H^T}$.
- Localizar que padrão de erro $\mathbf{e_i}$ corresponde a síndrome calculada.
- Corrigir a palavra recebida **r**, fazendo: $\mathbf{c} = \mathbf{r} + \mathbf{e_i}$.

Alguns códigos permitem uma abordagem mais simples e eficaz. Um exemplo simples de código de blocos com capacidade de correção de 1 erro é o chamado Código de Hamming [22].

3.2.3 Códigos Cíclicos

Os códigos cíclicos são uma subclasse dos códigos de blocos lineares, sendo muito utilizados por serem fáceis de codificar e decodificar. Além de possuírem uma estrutura matemática bem definida, podem ser facilmente implementados utilizando registradores de deslocamento ou lógica sequencial [24–26].

Um código cíclico, além de exibir a propriedade de linearidade dos códigos lineares, precisa exibir a propriedade cíclica, ou seja, *cada deslocamento cíclico de uma palavra-código é também uma palavra-código*. Por exemplo, se $\mathbf{u} = (u_o, u_1, ..., u_{n-1}, u_n)$ é uma palavra-código, o deslocamento cíclico pra direita de uma posição resultando em $\mathbf{u} = (u_n, u_o, u_1, ..., u_{n-1})$ também é uma palavra-código.

Uma das vantagens dos códigos cíclicos está no fato de que as palavras-códigos são representadas na forma de polinômios, sendo algebricamente mais fáceis de analisar e implementar. Esses polinômios são definidos em um Campo de Galois $GF(2^m)$, onde m é o número de bits.

A representação polinomial é feita da seguinte maneira. Considere um vetor $\mathbf{c}$ como sendo uma palavra-código definida por,

$$\mathbf{c} = (c_0, c_1, c_2, ..., c_{n-1}), \tag{3.13}$$

na representação polinomial, os componentes do vetor são os coeficientes do polinômio (0s ou 1s), que pertencem ao Campo de Galois no qual o polinômio foi definido, ou seja,

$$c(X) = c_0 + c_1X + c_2X^2 + ... + c_{n-1}X^{n-1}, \tag{3.14}$$

sendo o grau do polinômio até *n-1*.

A Tabela 3.2 mostra alguns exemplos de representação polinomial de palavras-código.

As operações de adição, multiplicação e divisão entre polinômios pertencentes a um Campo de Galois $GF(2^m)$ são feitas em aritmética *módulo-2*. Uma descrição mais detalhada de como é feita a aritmética *módulo-2* e a construção de Campos de Galois pode ser encontrada em [24].

O deslocamento cíclico pra direita de i posições de um vetor $\mathbf{c}$ pode ser representado na forma polinomial como:

$$c^{(i)}(X) = c_{n-i} + c_{n-i+1}X + ... + c_{n-i-1}X^{n-1}, \tag{3.15}$$

Tabela 3.2: Representação polinomial de palavras-código.

Palavras-código	**Polinômio c(X)**
0 0 0 0	0
1 0 1 0	$1 + X^2$
0 1 0 1	$X + X^3$
1 1 1 1	$1 + X + X^2 + X^3$

e a relação entre $c^{(i)}(X)$ e o polinômio $c(X)$ é tal que $c^{(i)}(X)$ é o resto da divisão de $X^i c(X)$ por $(X^n + 1)$, ou seja,

$$X^i c(X) = q(X)(X^n + 1) + c^{(i)}(X). \tag{3.16}$$

De outra forma pode-se dizer que,

$$c^{(i)}(X) = X^i c(X) \bmod (X^n + 1), \tag{3.17}$$

onde *mod* é a operação de *módulo* que se refere ao resto da divisão $X^i c(X)$ por $(X^n + 1)$.

3.2.3.1 Polinômio Gerador

O Polinômio Gerador $g(x)$ de um código cíclico (n,k) é o polinômio de menor grau entre os polinômios do código, e pode ser definido como,

$$g(X) = 1 + g_1 X + g_2 X + ... + g_{n-k-1} X^{n-k-1} + X^{n-k}, \tag{3.18}$$

no qual o grau n - k corresponde ao número de dígitos de paridade, e os coeficientes g_i são iguais a 0 ou 1.

Qualquer polinômio de um código é múltiplo do polinômio gerador, portanto, considerando uma mensagem $m(X)$, para codificá-la e obter o vetor código $c(X)$, basta multiplicar $m(X)$ por $g(X)$, ou seja,

$$c(X) = m(X)g(X) = (m_0 + m_1 X + ... + m_{k-1} X^{k-1})g(X). \tag{3.19}$$

Porém, essa codificação serve apenas para os códigos *não-sistemáticos*. Em um código *sistemático*, no qual o polinômio da mensagem deve permanecer de forma inalterada, para realizar a codificação é necessário seguir 3 passos [25, 26]:

1. Multiplica-se o polinômio da mensagem $m(X)$ por X^{n-k}

2. Obtêm-se o resto $b(X)$ da divisão de X^{n-k} por $g(X)$ que corresponde aos dígitos de paridade.

3. Adiciona-se X^{n-k} a $b(X)$ para obter o polinômio código $c(X)$.

Além disso, nos código sistemáticos nos quais a paridade é adicionada no final, é necessário inverter a ordem dos polinômios para serem lidos com o fator de maior ordem a esquerda, por exemplo, sendo $g(X) = 1 + X + X^3$, será considerado $g(X) = X^3 + X + 1$, apenas para melhor visualização.

Para exemplificar o processo de codificação dos códigos cíclicos pode-se seguir o exemplo a seguir.

Considere um código cíclico $(7, 4)$ com polinômio gerador $g(X) = X^3 + X + 1$ e a mensagem a ser codificada como sendo $m(X) = (0101) = X^2+1$. Seguindo os passos anteriores primeiro calculamos $X^3m(X) = X^5 + X^3$ e depois dividimos pelo polinômio $g(X)$:

$$\begin{array}{rl} \cancel{X^5} + \cancel{X^3} & \mid X^3 + X + 1 \\ \underline{\cancel{X^5} + \cancel{X^3} + X^2} & X^2 \\ X^2 = b(X) & \end{array} \tag{3.20}$$

obtendo o resto $b(X) = X^2$. Depois fazemos,

$$c(X) = X^3m(X) + b(X) = X^5 + X^3 + X^2 = (0101100).$$

Mantendo a mensagem (0101) inalterada e a paridade (100) adicionada no final.

3.2.3.2 Polinômio de Verificação de Paridade

O polinômio de verificação de paridade de um código cíclico gerado por um polinômio gerador $g(X)$ é um polinômio de grau k definido por:

$$h(X) = 1 + h_1X + h_2X + ... + h_{k-1}X^{k-1} + X^k. \tag{3.21}$$

Ambos os polinômios $g(X)$ e $h(X)$ são fatores do polinômio $X^n + 1$, podendo ser expressos por

$$g(X)h(X) = X^n + 1. \tag{3.22}$$

3.2.3.3 Matrizes Geradoras e de Verificação de Paridade

Um código cíclico *(n,k)* pode ser representado por uma matriz geradora na qual as linhas correspondem as palavras-código geradas pelo polinômio gerador $g(X)$ e os seus $k - 1$

deslocamentos cíclicos [28]:

$$\mathbf{G} = \begin{bmatrix} g(X) \\ Xg(X) \\ X^2g(X) \\ \vdots \\ X^{k-1}g(X) \end{bmatrix}. \tag{3.23}$$

Porém, essa matriz geradora não é da forma sistemática. Para obter a matriz na forma sistemática algumas operações com as linhas da matriz podem ser feitas. Por exemplo, considere a matriz geradora de um código cíclico $(7, 4)$ gerado com o polinômio $g(X) = 1 + X + X^3$,

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Se a primeira linha for adicionada a terceira linha, e a soma das duas primeiras linhas for adicionada a quarta linha, obtemos a matriz geradora na forma sistemática:

$$\mathbf{G}' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

A matriz de verificação de paridade pode ser obtida com o polinômio recíproco de $h(X)$, definido como:

$$h_r(X) = X^k h(X^{-1}) = h_k + h_{k-1}X + h_{k-2}X^2 + ... + h_0X^k. \tag{3.24}$$

E analogamente a matriz geradora, a matriz de verificação de paridade pode ser representada por:

$$\mathbf{H} = \begin{bmatrix} h_r(X) \\ Xh_r(X) \\ X^2h_r(X) \\ \vdots \\ X^{n-k-1}h_r(X) \end{bmatrix}. \tag{3.25}$$

3.2.3.4 Síndrome e Decodificação

Como visto na decodificação por síndromes dos códigos de blocos na Seção 3.2.2.3, o primeiro passo para realizar a decodificação é o cálculo da síndrome. O mesmo acontece para os códigos cíclicos. Consideramos que uma palavra-código $c(X)$ foi transmitida, e a palavra recebida, que pode ou não conter erros, seja representada pelo polinômio:

$$r(X) = r_0 + r_1X + r_2X^2 + ... + r_{n-1}X^{n-1}. \tag{3.26}$$

Para calcular a síndrome, dividimos o polinômio por $g(x)$, admitindo $q(X)$ como o quociente e $s(X)$ como o resto da divisão, obtendo,

$$r(X) = q(X)g(X) + s(X). \tag{3.27}$$

O resto da divisão $s(X)$ é o chamado polinômio da *síndrome* com grau $(n-k-1)$ ou menor. Para se detectar erros através da *síndrome*, é verificado se o resto $s(X)$ da divisão é igual a zero, se for, então significa dizer que o polinômio da palavra recebida é uma palavra-código e logo erros não ocorreram, porém se o polinômio *síndrome* for diferente de zero, então a palavra recebida contém erros de transmissão. Essa é a detecção que os códigos de *verificação de redundância cíclica* ou CRC *(Cyclic Redundancy Check)* realizam. Os códigos CRC são usados na codificação do ADSL e serão explicados mais adiante.

Depois de calculada a síndrome, a decodificação é feita do mesmo modo do processo de decodificação dos códigos de blocos, construindo uma tabela entre as síndromes e os padrões de erro. Lembrando que, a síndrome de um polinômio da palavra recebida também é a síndrome do erro correspondente [25], ou seja, dado que a palavra recebida contendo erros pode ser expressa como,

$$r(X) = c(X) + e(X), \tag{3.28}$$

onde $c(X)$ é a palavra transmitida e $e(X)$ o polinômio do erro, e de maneira equivalente, o polinômio erro pode ser expresso como,

$$e(X) = r(X) + c(X). \tag{3.29}$$

Substituindo as Equações 3.19 e 3.27 na Equação 3.29, obtém-se,

$$e(X) = (m(X) + q(X))g(X) + s(X), \tag{3.30}$$

provando que a síndrome $s(X)$ também é síndrome do polinômio erro $e(X)$.

A Tabela 3.3 mostra os padrões de erros e suas respectivas síndromes para um código cíclico $(7, 4)$ gerado com o polinômio gerador $g(X) = 1 + X + X^3$, com distância mínima de Hamming $d_{min} = 3$, podendo, portanto, corrigir um erro.

Tabela 3.3: Tabela de decodificação para o Código de Hamming *(7,4)*.

Padrão de erro $e(X)$	**Síndrome** $s(X)$	**Vetor Síndrome**
$e_6(X) = X^6$	$s(X) = 1 + X^2$	(101)
$e_5(X) = X^5$	$s(X) = 1 + X + X^2$	(111)
$e_4(X) = X^4$	$s(X) = X + X^2$	(011)
$e_3(X) = X^3$	$s(X) = 1 + X$	(110)
$e_2(X) = X^2$	$s(X) = X^2$	(001)
$e_1(X) = X^1$	$s(X) = X$	(010)
$e_0(X) = X^0$	$s(X) = 1$	(100)

Lembrando que a notação pode ser modificada para leitura do fator de maior expoente do polinômio como sendo o mais a esquerda, para garantir que a codificação siga o padrão de leitura no qual a informação vem primeiro e depois a paridade.

Entre os mais importantes códigos cíclicos estão os códigos CRC, que apenas detectam erros, os códigos BCH *(Bose-Chaudhuri-Hocquenghem)* [29] que são códigos binários capazes de corrigir até t erros e os códigos Reed-Solomon, que são códigos não-binários que operam em múltiplos bits, e que será apresentado na próxima seção.

3.2.4 Códigos Reed-Solomon

Os códigos Reed-Solomon surgiram em 1960, com a publicação de um artigo [30] por Irving Reed e Gus Solomon que descrevia uma nova classe de códigos corretores de erros, hoje conhecidos como Códigos Reed-Solomon. Atualmente, os códigos Reed-Solomon são de grande utilidade e de extrema importância para diversas aplicações, como CD's, comunicação sem fio, TV digital, comunicações via satélite e modems banda larga.

Os códigos Reed-Solomon (RS) são códigos cíclicos não-binários capazes de detectar e corrigir erros, pertencentes a uma subclasse dos códigos BCH que são códigos cíclicos binários. Diferentemente dos códigos BCH, os códigos RS operam em múltiplos bits sobre um campo finito, denominado Campo de Galois $GF(2^m)$, onde m é o número de bits, podendo ser qualquer valor maior ou igual a 2.

Como na descrição dos códigos lineares e dos códigos cíclicos nas Seções 3.2.2 e 3.2.3, respectivamente, os códigos Reed-Solomon na forma sistemática são especificados como $RS(N, K)$, onde N é definido como o número total de símbolos codificados e K é o número de

símbolos a ser codificados. A diferença $(N-K)$ é o número de símbolos de paridade que devem ser adicionados a mensagem permitindo que os erros possam ser corrigidos. E a capacidade de correção de um código RS, ou seja, a quantidade de erros que podem ser corrigidos por palavra-código é chamada de t e definida como sendo: $t = \frac{N-K}{2}$ [31].

Um código $RS(N, K)$ com capacidade de correção t e com símbolos pertencentes a um $GF(2^m)$, possui, portanto, os seguintes parâmetros [24, 26]:

- Tamanho da Palavra-código: $N = 2^m - 1$
- Número de símbolos de paridade: $N - K = 2t$
- Distância mínima: $d_{min} = N - K + 1$
- Capacidade de correção de erros: t erros por palavra-código.

As palavras-código nos códigos RS são representadas na forma de polinômios, assim como na descrição dos códigos cíclicos. Os coeficientes dos polinômios são definidos em um Campo de Galois $GF(2^m)$, portanto, para entender a codificação e decodificação de um código RS é necessário entendermos primeiramente como é definido um Campo de Galois.

De forma resumida, para cada número primo, $p = 2^m$, existe um Campo finito denominado $GF(p)$ que contém p elementos. Os p elementos pertencentes ao campo são representados por 0 ou 1 e por um elemento primitivo de $GF(2^m)$, representado pela letra β, onde cada elemento pode ser representado por uma potência de β. Os elementos do Campo finito de 2^m elementos podem então ser definidos como [31]:

$$GF(2^m) = \{0, 1, \beta^1, \beta^2, \beta^3, ..., \beta^{2^m-2}\}. \tag{3.31}$$

Cada um dos 2^m elementos do Campo podem ser representados por um polinômio de grau $m-1$ ou menos. Além disso, os elementos do Campo de Galois são gerados através de um chamado *polinômio primitivo*. Um polinômio primitivo $p(X)$ é um polinômio de grau m que tem o elemento primitivo β como raiz, ou seja $p(\beta) = 0$ [24].

Como exemplo de construção de um Campo de Galois considere um Campo de Galois $GF(2^3)$, com $2^3 = 8$ elementos. Usando o polinômio primitivo $p(X) = X^3 + X^2 + 1$, pode-se obter a representação polinomial dos elementos fazendo, X mod $p(X)$, resultando na segunda coluna da Tabela 3.4.

A terceira coluna da Tabela 3.4 representa o polinômio na forma binária gerando a palavra-código. A representação polinomial de palavras-código pode ser encontrada na Tabela 3.2.

Tabela 3.4: Campo de Galois $GF(2^3)$ gerado pelo polinômio primitivo $p(X) = X^3 + X^2 + 1$.

Potência de β^i	**Polinômio em $X^i \bmod p(x)$**	**Palavra-código**
0	0	000
$\beta^0 = 1$	1	001
β^1	X	010
β^2	X^2	100
β^3	$X^2 + 1$	101
β^4	$X^2 + X + 1$	111
β^5	$X + 1$	011
β^6	$X^2 + X$	110

Um exemplo simples da geração de um elemento no $GF(2^3)$, pode ser visto considerando, por exemplo, o cálculo de β^3.

$\beta^3 = X^3 \bmod (X^3 + X^2 + 1) = X^2 + 1$. (resto da divisão)

A prova do polinômio primitivo, assim como um exemplo de como funciona a operação de soma no Campo de Galois pode ser visto a seguir:

$$p(\beta) = X^3 + X^2 + 1 = \beta^3 + \beta^2 + 1 = X^2 + 1 + X^2 + 1 = 0. \tag{3.32}$$

Uma lista com diversos Campos de Galois para $3 < m < 10$ pode ser encontrado em [26].

3.2.4.1 Codificação

Na codificação dos Códigos RS na forma sistemática, como já visto anteriormente, os bytes ou símbolos de redundância chamados aqui de $R = N - K$ são acrescentados aos bytes de informação K gerando um palavra-código RS de tamanho $N = K + R$. Os bytes de redundância ou paridade são o resto da divisão polinomial da mensagem $M(X)$ de tamanho K, por um polinômio gerador $G(X)$. A palavra-código RS na forma polinomial pode ser expressa como:

$$C(X) = M(X)X^R \bmod G(X). \tag{3.33}$$

O polinômio gerador para um código RS pode ser expresso na forma [26]:

$$G(X) = g_0 + g_1X + g_2X^2 + ... + g_{2t-1}X^{2t-1} + g_{2t}X^{2t}, \tag{3.34}$$

ou como:

$$G(X) = \prod_{i=1}^{R}(X + \beta^i), \tag{3.35}$$

no qual o grau do polinômio é igual ao número de paridade $2t$ ou R.

Como exemplo de polinômio gerador, será considerado um código $RS(7, 5)$, no qual $N = 7$, $K = 5$ e $R = 2t = 2$. O polinômio gerador pode ser calculado através da Equação 3.35, resultando em:

$$\begin{aligned} G(X) = (X + \beta)(X + \beta^2) &= X^2 + \beta X + \beta^2 X + \beta^3 \\ &= X^2 + \beta^6 X + \beta^3. \end{aligned} \tag{3.36}$$

Segue um exemplo para entender a codificação RS usando Campos de Galois.

Considere o código $RS(7, 5)$, com elementos do $GF(2^3)$ e a mensagem a ser transmitida com tamanho $K - 5$ como, (001 101 111 010 011), e na forma polinomial usando a Tabela 3.4: $M(X) = X^4 + \beta^3 X^3 + \beta^4 X^2 + \beta X + \beta^5$. Assim:

$$\begin{aligned} X^R M(X) &= X^2 M(X) \\ &= X^2[X^4 + \beta^3 X^3 + \beta^4 X^2 + \beta X + \beta^5] \\ &= X^6 + \beta^3 X^5 + \beta^4 X^4 + \beta X^3 + \beta^5 X^2. \end{aligned}$$

Agora fazendo a divisão de $X^R M(X)$ pelo polinômio gerador da Equação 3.36.

$$\begin{array}{ll} \cancel{X^6} + \beta^3 X^5 + \beta^4 X^4 + \beta X^3 + \beta^5 X^2 & \mid X^2 + \beta^6 X + \beta^3 \\ \underline{\cancel{X^6} + \beta^6 X^5 + \beta^3 X^4} & X^4 + \beta^5 X^3 + \beta^3 X^2 + \beta^2 X + \beta^4 \\ \quad \cancel{\beta^5 X^5} + \beta X^4 + \beta X^3 + \beta^5 X^2 & \\ \quad \underline{\cancel{\beta^5 X^5} + \beta^4 X^4 + \beta X^3} & \\ \quad\quad \cancel{\beta^3 X^4} + \beta^5 X^2 & \\ \quad\quad \underline{\cancel{\beta^3 X^4} + \beta^2 X^3 + \beta^6 X^2} & \\ \quad\quad\quad \cancel{\beta^2 X^3} + \beta^3 X^2 & \\ \quad\quad\quad \underline{\cancel{\beta^2 X^3} + \beta X^2 + \beta^5 X} & \\ \quad\quad\quad\quad \cancel{\beta^4 X^2} + \beta^5 X & \\ \quad\quad\quad\quad \underline{\cancel{\beta^4 X^2} + \beta^3 X + 1} & \\ \quad\quad\quad\quad R(X) = \beta^6 X + 1. & \end{array} \tag{3.37}$$

Lembrando que a operação de adição na divisão polinomial usando Campos de Galois é feita como no exemplo da Equação 3.32, usando por exemplo o GF definido na Tabela 3.4.

O polinômio $R(X)$ representando a paridade é então adicionado no final do polinômio mensagem $X^R M(X)$, resultando no polinômio da mensagem codificada $C(X)$:

$$C(X) = X^6 + \beta^3 X^5 + \beta^4 X^4 + \beta X^3 + \beta^5 X^2 + \boldsymbol{\beta^6 X} + \mathbf{1} \tag{3.38}$$

Resultando na palavra código: (001 101 111 010 011 110 001).

3.2.4.2 Decodificação

A decodificação dos códigos RS é uma operação bem mais complexa que a codificação, passando por várias etapas. Mais precisamente, o processo de decodificação pode ser dividido em 4 etapas:

1. Calcular as Síndromes:

   O primeiro passo no processo de decodificação é o cálculo das síndromes. Considera-se que uma palavra código $C(X)$ foi codificada e transmitida, e que a palavra recebida podendo conter erros é representada pelo polinômio $r(X)$, para calcular as síndromes é necessário substituir no polinômio recebido $r(X)$, a variável X pelas raízes $\beta^i, i = 0, 1, 2, ..., 2t$, como definido na seguinte Equação [31]:

   $$S_i = r(X)|_{X=\beta^i} = r(\beta^i) \qquad i = 1, 2, ..., 2t. \tag{3.39}$$

   O número de síndromes calculadas é igual a paridade $2t$. Se o cálculo das síndromes resultar em zero, então a mensagem não contém erros.

2. Determinar o polinômio localizador de erros $\sigma_{BM}^{(\mu)}(X)$ (algoritmo de Berlekamp-Massey):

   O cálculo do polinômio localizador de erros $\sigma_{BM}^{(\mu)}(X)$ é realizado através do algoritmo de Berlekamp-Massey [32, 33]. O algoritmo de Berlekamp-Massey pode ser implementado através de uma Tabela, como mostrado na Tabela 3.5.

   Segundo [24], a Tabela é preenchida linha por linha para cada valor de $\mu+1$. A constante l_μ é o grau do polinômio $\sigma_{BM}^{(\mu)}(X)$ e a quantidade d_μ é chamada de μ-ésima discrepância. Os valores das linhas $\mu + 1$ são obtidos através das informações da linha μ, avaliando o valor de d_μ, ou seja, o polinômio $\sigma_{BM}^{(\mu+1)}(X)$ na iteração $\mu + 1$ é calculado como função do polinômio $\sigma_{BM}^{(\mu)}(X)$ da μ-ésima iteração, através das seguintes condições:

   - Se $d_\mu = 0$, então $\sigma_{BM}^{(\mu+1)}(X) = \sigma_{BM}^{(\mu)}(X)$ e $l_{\mu+1} = l_u$.
   - Se $d_\mu \neq 0$, é necessário definir o valor de ρ, no qual ρ é o passo anterior em que $d_\rho \neq 0$. Então:

     $$\sigma_{BM}^{(\mu+1)}(X) = \sigma_{BM}^{(\mu)}(X) + d_\mu d_\rho^{-1} x^{(\mu+\rho)} \sigma^{(\rho)}(X), \tag{3.40}$$

Tabela 3.5: Tabela do algoritmo Berlekamp-Massey para determinação do polinômio localizador de erros $\sigma_{BM}^{(\mu)}(X)$ [24].

μ	$\sigma_{BM}^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	S_1	0	0
1				
⋮				
2t				

$$l_{\mu+1} = \max(l_\mu, l_\rho + \mu - \rho). \tag{3.41}$$

Para ambos os casos o valor de d_μ é definido por:

$$d_{\mu+1} = S_{\mu+2} + \sigma_1^{(\mu+1)} S_{\mu+1} + \ldots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{\mu+2-l_{\mu+1}}. \tag{3.42}$$

O procedimento é aplicado até que se encontre o polinômio $\sigma_{BM}^{(2t)}(X)$, que será adotado como o polinômio localizador de erros final.

Exemplos da decodificação de códigos RS usando o algoritmo de Berlekamp-Massey podem ser encontrados em [24, 26].

3. Determinar as raízes do polinômio localizador de erros (*Chien Search*):

   Após encontrado o valor do polinômio localizador de erros $\sigma_{BM}^{(\mu)}(X)$, as suas raízes determinam onde os erros estão localizados na palavra-código recebida. Para isso, utiliza-se um método denominado "Chien Search", que nada mais é do que a substituição de todos os 2^m possíveis símbolos do Campo de Galois no polinômio localizador de erros $\sigma_{BM}^{(\mu)}(X)$. Se o resultado for igual a zero, então esse determinado símbolo é raiz do polinômio.

4. Calcular a magnitude dos erros (algoritmo de *Forney*):

   O último passo é o calculo da magnitude dos erros. Com os passos anteriores já determinou-se a localização dos erros, porém é necessário saber o valor dos erros para que ocorra a decodificação. Para isso, utiliza-se o algoritmo de Forney, que consiste primeiro na definição do polinômio $Z(X)$ [24, 26]:

$$Z(X) = 1 + (S_1 + \sigma_1)X + (S_2 + \sigma_1 S_1 + \sigma_2)X^2 + \ldots + (S_\tau + \sigma_1 S_{\tau-1} + \sigma_2 S_{\tau-2} + \ldots + \sigma_\tau)X^\tau, \tag{3.43}$$

depois no cálculo da magnitude dos erros nas posições β_i^{-1}:

$$e_{j_i} = \frac{Z(\beta_i^{-1})}{\prod_{k=1,k\neq i}^{\tau}(1+\beta_k\beta_i^{-1})}. \tag{3.44}$$

De posse da posição e do valor dos erros é possível definir o polinômio de erros que será somado com a palavra recebida, resultando finalmente, na palavra transmitida.

Caso a quantidade de erros de uma palavra recebida for maior do que a capacidade de correção de erros do código RS, a decodificação não pode ser realizada e nenhum erro da palavra recebida é corrigido, ou seja, a palavra permanece inalterada após passar pela decodificação do RS.

Shortened Reed-Solomon

Em alguns projetos de sistemas, há a necessidade de se usar códigos RS de tamanhos menores do que os usados naturalmente, para isso existe uma técnica denominada *shortening*, que pode ser aplicada para qualquer código cíclico. *Shortening* um código RS [26, 34] consiste em diminuir o número de bytes de informação K e o tamanho da palavra-código N, a partir de um código RS já existente. No entanto, mantendo o tamanho dos bytes de paridade $(N-K)$ iguais, para garantir que o código "shortened" tenha a mesma capacidade de correção, porém sem ser um código cíclico.

A codificação e decodificação dos códigos *shortened* RS é realizada acrescentando-se zeros na palavra-código para garantir que o tamanho da palavra-código N seja igual a $2^m - 1$. Por exemplo, considera-se um código RS(255,241), com 14 bytes de paridade e capacidade de correção igual a 7 bytes. Um código *shortened* RS poderia ser RS(143,129), com 14 bytes de paridade e a mesma capacidade de correção. Antes da palavra-código ser codificada, 114 zeros são adicionados aos 129 bytes de informação, garantindo que o número de bytes de informação seja igual a 241. No final da codificação, após os bytes de paridade serem calculados, os 112 zeros são retirados, resultando em uma palavra-código de tamanho N igual a 143. Na decodificação, o mesmo número de zeros precisam ser acrescentados e depois retirados.

3.2.5 *Interleaver*

O *interleaver* é uma técnica usada na proteção contra erros em rajada (erros em *burst*) que tem como objetivo embaralhar uma sequência de dados. No ADSL, os erros causados por ruído impulsivo são concentrados em rajadas de erros, onde a duração do ruído pode ocasionar numa quantidade de bytes errados em uma palavra-código muito maior do que a

capacidade de correção do código Reed-Solomon, não permitindo a correção. A combinação do *interleaver* com o Reed-Solomon aumenta a possibilidade de correção desses erros, pois o *interleaver* rearruma os bytes das palavras-código em uma ordem diferente, permitindo que os bytes danificados se espalhem entre as palavras-código e possam ser corrigidos pelo Reed-Solomom [8]. No receptor, o desembaralhamento é realizado por um *de-interleaver* que recupera a ordem dos bytes nas palavras-código.

Um dos tipos de *interleavers*, o *interleaver* de bloco, por exemplo, formata a mensagem em uma matriz $m \times n$, onde cada linha da matriz consiste em uma palavra código de tamanho N. Os bytes ou símbolos são escritos nas linhas da matriz e lidos nas colunas da matriz.

Como exemplo, considera-se um código RS(7,4), com capacidade de correção $t = 2$, e a sequência a ser enviada como sendo:

(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21).

A sequência é escrita na matriz como:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 15 & 16 & 17 & 18 & 19 & 20 & 21 \end{bmatrix}.$$

E lida nas colunas da matriz, resultando na sequência após o *interleaver*:

(1 8 15 2 9 16 3 10 17 4 11 18 5 12 19 6 13 20 7 14 21).

Considera-se agora que a sequência foi corrompida por erros em rajada, resultando em 3 bytes errados:

(1 8 15 2 9 16 3 10 **17 4 11** 18 5 12 19 6 13 20 7 14 21).

No *de-interleaver* a mensagem é recuperada para a ordem original, resultando em:

$$\begin{bmatrix} 1 & 2 & 3 & \mathbf{4} & 5 & 6 & 7 \\ 8 & 9 & 10 & \mathbf{11} & 12 & 13 & 14 \\ 15 & 16 & \mathbf{17} & 18 & 19 & 20 & 21 \end{bmatrix}.$$

Se a sequência não for embaralhada o número de erros supera a capacidade de correção de erros (t=2), fazendo com que os erros não sejam corrigidos, porém com o *interleaver* cada palavra-código agora possui apenas um erro, possibilitando que os erros possam ser corrigidos pelo Reed-Solomon.

As técnicas de *interleaver* possuem a desvantagem de aumentar a latência do sistema. Nos *interleavers* de bloco, por exemplo, é preciso esperar que todo o bloco seja preenchido antes do bloco poder ser desembaralhado.

O ADSL utiliza um outro tipo de *interleaver* denominado *interleaver* convolucional que será explicado na Seção 4.6.

Capítulo 4

Codificação e Controle de erro em Sistemas ADSL

No ADSL, como em qualquer sistema de comunicação, a transmissão de dados está sujeita a ruídos, distorções e interferências que podem degradar a informação sendo transmitida. Portanto, se faz necessário o uso de controle de erro e codificação, a fim de detectar e corrigir esses erros. Para um sistema ADSL, as especificações do funcionamento da codificação de canal podem ser encontradas no padrão ADSL G.992.1 [3].

O padrão ADSL G.999.2.1 descreve o transmissor de um modem ADSL, especificando os requerimentos necessários para permitir alta velocidade de transmissão de dados em um par trançado metálico entre a central (ATU-C) e o usuário final (ATU-R) e garantir a interface e interoperabilidade adequada. As especificações incluem requisitos obrigatórios, recomendações e opções de configuração como, por exemplo, especificações da camada física, protocolos, técnicas de transmissão, controle de erro, entre outros.

4.1 Esquema de um modem ADSL

O diagrama de blocos do transmissor e do receptor de um modem ADSL é mostrado na Fig 4.1. A Figura 4.2 mostra o diagrama de blocos do transmissor ADSL recomendado no padrão [3].

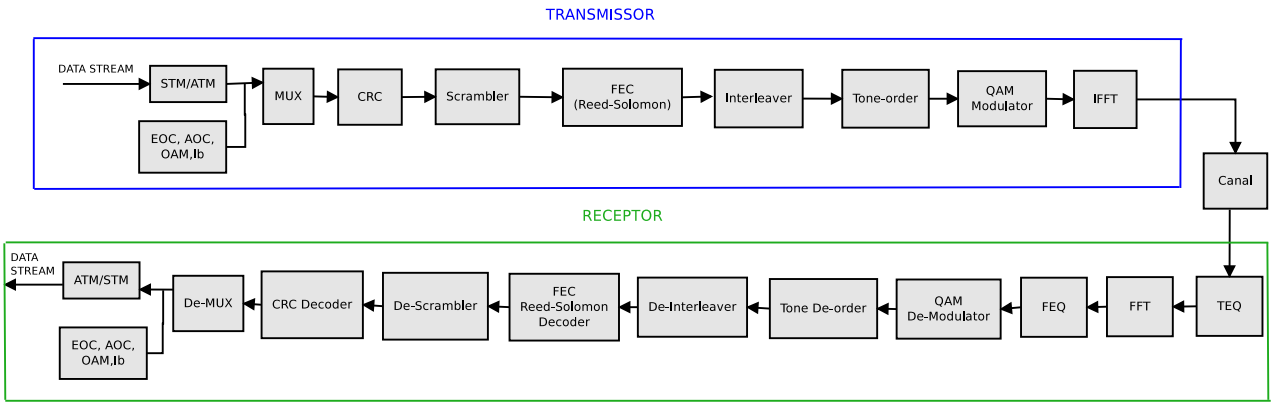


Figura 4.1: Diagrama de blocos do transmissor e receptor de um modem ADSL.

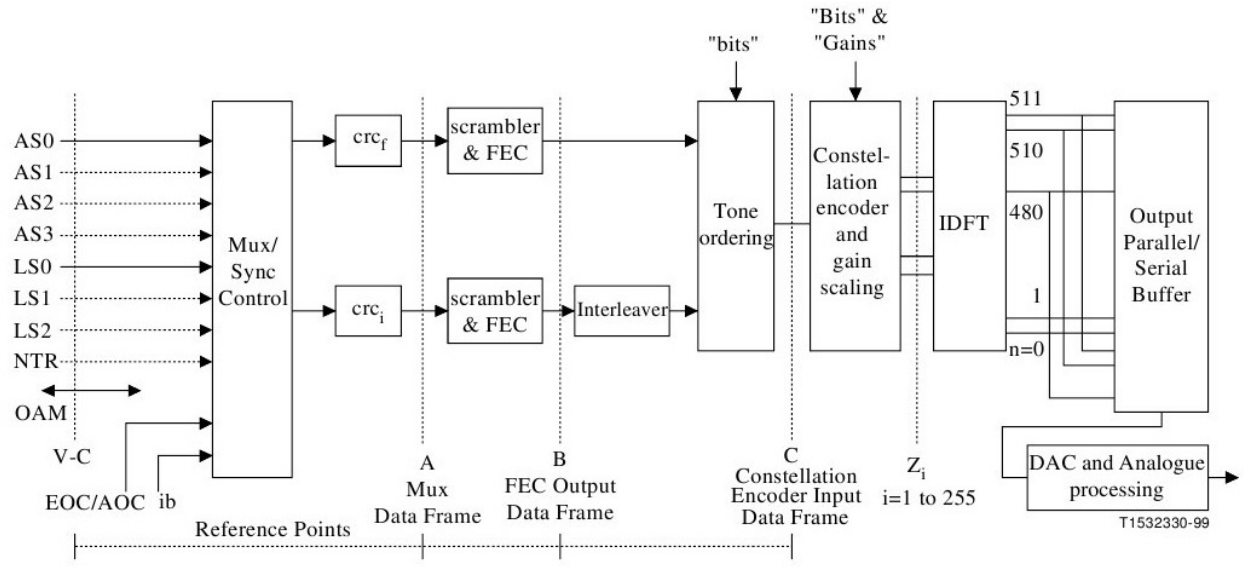


Figura 4.2: Diagrama de blocos do transmissor ADSL no ATU-C [3].

4.2 Estrutura dos *Frames*

Os bits de informação recebidos na entrada do modem são agrupados em uma estrutura denominada *superframes*, como mostrada na Figura 4.3. Cada *superframe* é compostos por 68 *frames* (0 a 67) e um símbolo adicional de sincronização inserido pelo modulador DMT para determinar os limites do *superframe*, totalizando 69 *frames*, com duração total de 17 ms. Cada *frame* é um grupo de bits que será codificado e modulado para um símbolo DMT, possuindo duração de 250 μs. Esses *frames* são compostos por dois *buffers*: *fast data buffer* e *interleaved data buffer*, que se diferem pela presença do *interleaver* no *interleaved data buffer*, como pode

ser visto do diagrama de blocos da Figura 4.2. O *fast buffer* é caracterizado por possuir pouca latência, porém com maior taxa de erro, sendo mais adequado para transmissão de voz que requer pouca latência e é relativamente tolerante a erros. Já o *interleaved buffer* possui maior latência devido a presença do *interleaver*, porém com menor taxa de erro, e é mais adequado para aplicações com vídeos que são caracterizadas por serem tolerantes a atrasos mas muito intolerantes a erros de transmissão [35].

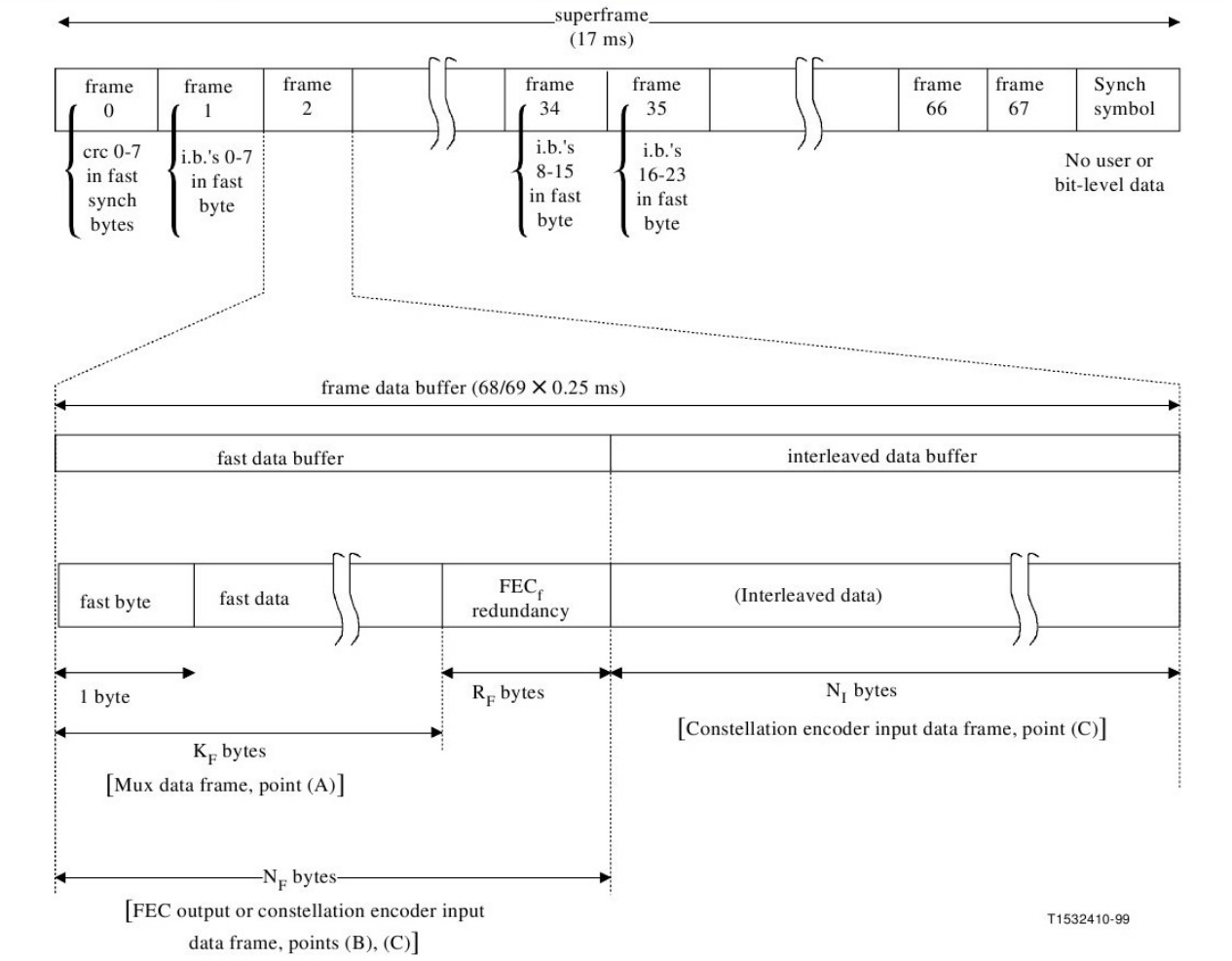


Figura 4.3: Estrutura de um *Superframe* ADSL [3].

De acordo com [3], no *fast buffer* o primeiro byte do *frame* é denominado de *fast byte*, no qual no *frame* 0, o *fast byte* carrega os 8 bits de CRC calculados e nos *frames* 1, 34 e 35 carrega bits indicadores de informação. Nos demais *frames*, os *fast bytes* são usados para EOC (*embedded operations channel*) ou para controle de sincronização. No *interleaved buffer*, cada *frame* é composto de um *sync byte* como o primeiro byte do *frame*, no qual no *frame* 0, esse *sync byte* carrega o CRC calculado para o *superframe* anterior, e nos outros *frames* (1 a 67),

esse byte é usado para controle de sincronização ou para controle de *overhead* (AOC).

A estrutura do *fast data buffer* é mostrada na Figura 4.4.

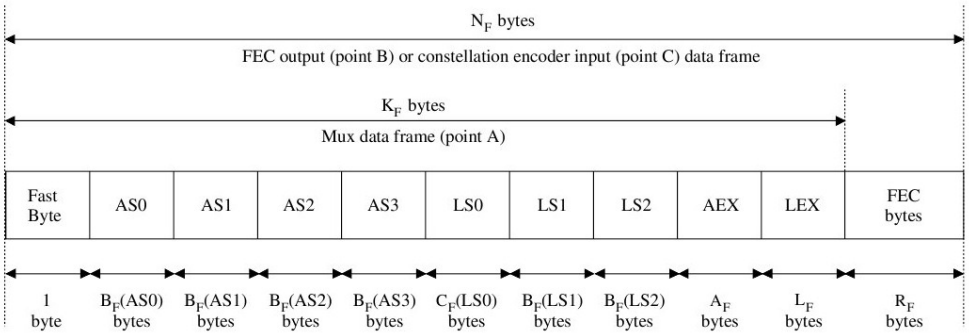


Figura 4.4: *Fast data buffer* [3].

O *bitstream* de informação (AS0, AS1, AS2, AS3, LS0, LS1, LS2, AEX e LEX) junto com o *fast byte* são armazenados nos K_F bytes que representam o ponto de referência A "mux data frame" da Figura 4.2, no qual os *frames* possuem ainda só a informação e o byte de CRC. Os R_F bytes de redundância são adicionados pelo codificador Reed-Solomon no FEC produzindo o "FEC Output data frame" no ponto de referência B, com tamanho igual a $N_F = K_F + R_F$. Como no *fast buffer* não há a presença do *interleaver*, o *frame* será enviado para o *Tone Order* formando o "constelation encoder input data frame" com o mesmo formato do "FEC Output data frame".

Na Figura 4.5 é mostrada a estrutura do *interleaved data buffer*. No ponto de referência A da Figura 4.2, o *interleaved buffer* possui o mesmo formato do *fast buffer*, ou seja, cada "mux data frame" terá tamanho de K_I bytes. No ponto de referência B, deve-se levar em conta a quantidade S de "mux data frames" utilizada. A constante S determina o número de *frames* ou símbolos DMT que podem ser colocados em uma palavra-código RS (S é igual a 1 para o *fast path*, ou seja, cada palavra-código RS possui apenas um *frame* ou símbolo DMT, porém esse valor pode variar para o *interleaver path*, podendo chagar até 16.)

Para cada "mux data frame" de tamanho K_I bytes são calculados os bytes de redundância. Os bytes de redundância de todos os S "mux data frames" são agrupados e adicionados no final do *frame* com tamanho igual R_I bytes, produzindo um *frame* de tamanho total igual a $N_{FEC} = S \times K_I + R_I$ bytes. Cada "FEC output data frame" é formado então com $N_I = N_{FEC}/S$ bytes. Os "FEC output data frames" são então *interleaved* por um determinado *interleaver depth* (que será melhor explicado na Seção 4.6), permanecendo com o mesmo tamanho, apenas com os bytes agora embaralhados entre *frames*, passando então para

o *Tone Order* formando o "constelation encoder input data frame" .

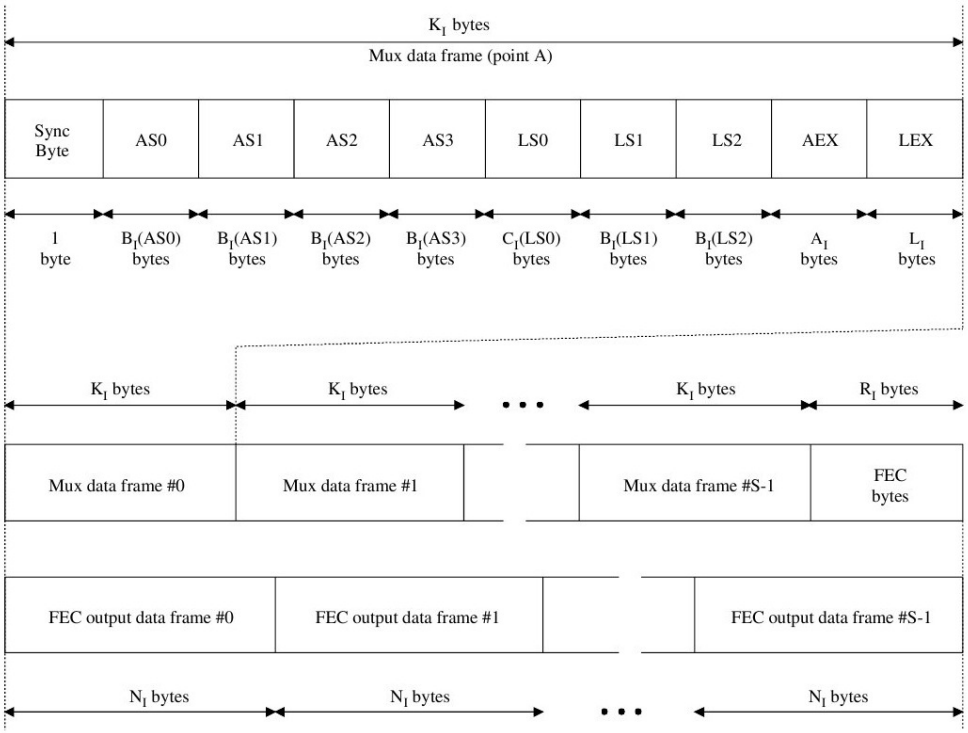


Figura 4.5: *Interleaved data buffer* [3].

4.3 CRC

Os códigos de *Verificação de Redundância Cíclica* ou CRC *(Cyclic Redundancy Check)* são códigos para detecção de erros. O chamado CRC são os bits de paridade ou redundância adicionados a mensagem a ser enviada que garantem que os erros possam ser detectados na decodificação.

O bloco CRC é o primeiro dos grupos de blocos responsáveis pela codificação de canal no ADSL. No transmissor ADSL, de acordo com o diagrama de blocos da Figura 4.2, dois CRCs são calculados para cada *superframe*, um para o *fast data buffer* e outro para o *interleaver data buffer*, contendo 8 bits cada. O CRC é o resto da divisão polinomial entra a mensagem a ser enviada e um polinômio gerador definido, como mostrado na seguinte equação:

$$CRC(X) = M(X)X^8 \bmod G(X), \tag{4.1}$$

na qual $M(X)$ é o polinômio da mensagem e representa os k bytes da mensagem a ser enviada,

definida por:

$$M(X) = m_0 X^{k-1} + m_1 X^{k-2} + ... + m_{k-2} X + m_{k-1}, \tag{4.2}$$

$G(X)$ é o polinômio gerador especificado para o ADSL com 8 bits [3]:

$$G(X) = X^8 + X^4 + X^3 + X^2 + 1, \tag{4.3}$$

e 8 é o número de bits de CRC.

Na Tabela 4.1 estão listados alguns dos polinômios geradores utilizados nos diferentes tipos de sistemas DSL.

Tabela 4.1: Polinômios Geradores de alguns códigos CRC usados nos padrões DSL [8].

Padrão	**m**	**Polinômio Gerador g(x)**
ADSL/ADSL2/ADSL2+/DMT VDSL1	8	$X^8 + X^4 + X^3 + X^2 + 1$
CRC-6: SHDSL/HDSL/HDSL2	6	$X^6 + X + 1$
QAM VDSL1	4	$X^4 + X + 1$
CRC-16: HDSL2/HDLC,IP	16	$X^{16} + X^{12} + X^5 + 1$

Depois de calculados, ambos os CRCs são transmitidos no primeiro byte do próximo *superframe* previamente reservado. Para detecção dos erros na decodificação, a mensagem recebida (mensagem $M(X)$ + CRC) é dividida novamente pelo polinômio gerador $G(X)$, se o resto da divisão for zero então a mensagem recebida não possui erros, como mostrado na seguinte equação:

$$(M(X) + CRC(X)) \bmod G(X) = 0. \tag{4.4}$$

4.4 *Scrambler*

Os *scramblers*, na transmissão de dados, são dispositivos usados para embaralhar uma sequência de bits. Uma das funções do *scrambler* é prevenir a transmissão de sequências de bits repetidos, como longas sequências de 1s ou de 0s, que podem ocasionar problemas de sincronização na transmissão. O embaralhamento é feito bit a bit usando um LFSR (*Linear Feedback Shift Register*) e um polinômio primitivo pré-definido. No ADSL, o *scrambler* embaralha os bits através do polinômio da Equação 4.5.

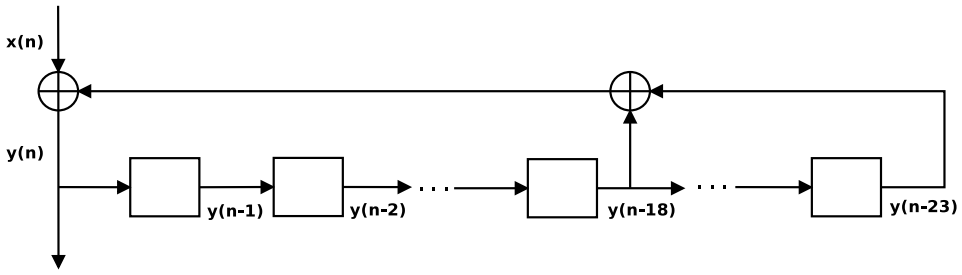
$$h(X) = 1 + X^{18} + X^{23}. \tag{4.5}$$

O diagrama de blocos da Figura 4.6 (a) mostra o LFRS usado para o *scrambler* no ADSL, onde $x(n)$ representa a sequência de bits na entrada do embaralhador. E a mensagem embaralhada $y(n)$ pode ser escrita como:

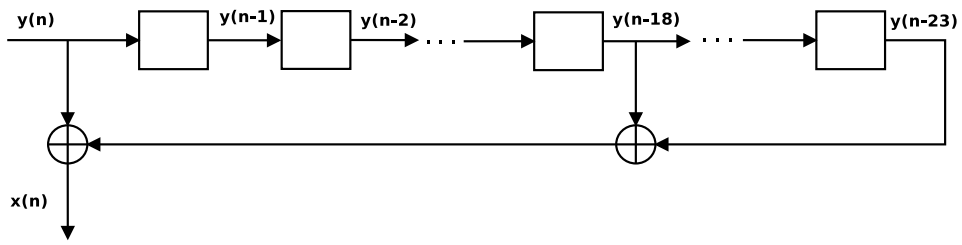
$$y(n) = x(n) + y(n-18) + y(n-23) \tag{4.6}$$

O *descrambler* é o responsável por desembaralhar os bits no receptor. A Figura 4.6 (b) representa o LFRS para o *descrambler*, mostrando como a sequência embaralhado pode ser desembaralhada usando o mesmo polinômio. E a mensagem desembaralhada $x'(n)$, após o *descrambler* pode ser escrita como:

$$x'(n) = y(n) + y(n-18 + y(n-23). \tag{4.7}$$



(a) Scrambler.



(b) Descrambler.

Figura 4.6: *Scrambler* e *Descrambler*.

Esse *scrambler* é chamado de *scrambler* auto-sincronizador pois não necessitam de sincronização de *frames* ou de símbolos [8].

4.5 Reed Solomon

No ADSL, o FEC utiliza os códigos Reed-Solomon para detecção e correção dos erros em uma transmissão. Segundo o padrão do ADSL [3], na codificação R bytes de redundância

ou paridade devem ser anexados a K bytes de informação ou mensagem para formar um palavra-código RS com $N = K+R$ bytes. Os R bytes de redundância são calculados, seguindo a explicação na Seção 3.2.4, e através da Equação:

$$C(X) = M(X)X^R \bmod G(X). \tag{4.8}$$

O polinômio $C(X)$ representando a redundância é o resto da divisão de $M(X)X^R$ por $G(X)$, onde $M(X)$ é o polinômio da mensagem e representa os K bytes da mensagem a ser enviada, definido por:

$$M(X) = m_0X^{k-1} + m_1X^{k-2} + ... + m_{k-2}X + m_{k-1}. \tag{4.9}$$

E $G(X)$ é o polinômio gerador do código Reed-Solomon, com o índice i variando de $i = 0$ a R-1, ou seja o grau do polinômio é igual a R-1, diferentemente do polinômio gerador descrito na Equação 3.35:

$$G(X) = \prod_{i=0}^{R-1}(X + \beta^i). \tag{4.10}$$

No caso do ADSL, o número de bits é $m = 8$, portanto a aritmética é feita em um Campo de Galois $GF(2^8) = GF(256)$. O Campo de Galois $GF(256)$ é gerado através do seguinte polinômio primitivo:

$$p(X) = X^8 + X^4 + X^3 + X^2 + 1. \tag{4.11}$$

Portanto, o tamanho máximo de uma palavra-código RS é N = 255 bytes. Os possíveis números de bytes de paridade R, assim como os valores do *interleaver depth* (que será descrito na seção seguinte), e o número S de símbolos DMT ou *frames* por palavra-código RS, tanto para o *downstream* quanto para o *upstream* permitidos para o ADSL, são descritos na Tabela 4.2. Com exceção do *interleaver depth* que possui valores diferentes para *downstream* e *upstream*, os outros valores são os mesmos para as duas direções.

4.6 *Interleaver* Convolucional

No ADSL, o tipo de *interleaver* utilizado é o chamado *interleaver* convolucional. Um *interleaver* convolucional é constituído por um conjunto de registradores de deslocamento, cada um com um atraso fixo. De acordo com [3], cada um dos N bytes de uma palavra-código são atrasados por um valor que varia linearmente com o índice do byte, mais precisamente, os bytes são atrasados em $(D-1)\times i$ bytes, onde D é o chamado *interleaver depth* e $i = 0, 1, 2, ..., N-1$, o índice do byte.

Tabela 4.2: Valores de bytes de paridade, símbolos DMT por palavra-código e *interleaver depth* permitidos no ADSL, para *downstream* e *upstream* [3].

Parâmetros	**Fast buffer**	**Interleaved buffer**
Bytes de paridade por palavra código RS	$R_F = 0, 2, 4, 6, 8, 10, 12, 14, 16$	$R_I = 0, 2, 4, 6, 8, 10, 12, 14, 16$
Símbolo DMT por palavra código RS	$S = 1$	$S = 1, 2, 4, 8, 16$
Interleave depth	não usado	$D_D = 1, 2, 4, 8, 16, 32, 64$ $D_U = 1, 2, 4, 8$

Diferentemente dos *interleavers* de bloco, onde é necessário esperar que o bloco todo seja recebido antes de realizar a leitura, os *interleavers* convolucionais se mostram mais vantajosos, pois requerem metade ou menos da metade da memória necessária. Uma desvantagem está no fato de que os valores de N e D precisam ser co-primos, ou seja, possuírem maior fator comum igual a 1 [36]. Além disso, para o ADSL, os *interleavers depth* devem ser em potências de 2, fazendo com que os valores de N para serem co-primos necessitem ser par, porém quando N for ímpar é possível adicionar um chamado *dummy* byte no início da palavra-código antes de passar pelo *interleaver*, apenas para garantir o funcionamento. Na saída do *interleaver* esse *dummy* byte é retirado.

Consideremos palavras-código de tamanho $N = 7$ e *interleaver depth* $D = 4$, seguindo a Tabela 4.3, observamos que o byte 0 é recebido sem atraso. O byte 1 é recebido com um atraso de 3 bytes, ou seja é lido na posição 4. O byte 2 é recebido com um atraso de 6 bytes, ou seja é lido na posição 8, e assim sucessivamente. Resultando na ordenação dos bytes na saída do *interleaver* como mostrado na Tabela 4.3.

O *de-interleaver* convolucional realiza o processo inverso, recuperando a ordem dos bytes da palavra-código. Descrições do funcionamento do *interleaver* e *de-interleaver* convolucional podem ser encontradas em [8]. Uma descrição mais detalhada do funcionamento do *interleaver* e *de-interleaver* também pode ser encontrado em [37]. Para descrever o funcionamento do *de-interleaver* usaremos a descrição em [37], usando registradores e tabelas de permutação.

No *de-interleaver*, a sequência atrasada proveniente do *interleaver* é armazenada em um registrador de tamanho $N \times D$. O preenchimento do registrador segue uma tabela de permutação baseada nos atrasos fixos. A Figura 4.7 mostra um exemplo do processo de

Tabela 4.3: Bytes *interleaved* e *de-interleaved*

i	bytes	bytes *interleaved*	bytes *de-interleaved*
0	0	**0**	x
1	1	x	x
2	2	x	x
3	3	x	x
4	4	**1**	x
5	5	x	x
6	6	x	x
0	7	**7**	x
1	8	**2**	x
2	9	x	x
3	10	x	x
4	11	**8**	x
5	12	**3**	x
6	13	x	x
0	14	**14**	x
1	15	**9**	x
2	16	**4**	x
3	17	x	x
4	18	**15**	x
5	19	**10**	x
6	20	**5**	x
0	21	**21**	**0**
1	22	**16**	**1**
2	23	**11**	**2**
3	24	**6**	**3**
4	25	**22**	**4**
5	26	**17**	**5**
6	27	**12**	**6**
⋮	⋮	⋮	⋮

de-interleaver com $N = 7$ e $D = 4$, usando a sequência embaralhada proveniente do *interleaver* da Tabela 4.3. Cada palavra-código recebida é colocada na primeira coluna do registrador seguindo a tabela de permutação. Os elementos da coluna 1 são então deslocados para a coluna 2, assim como os da coluna 2 são deslocados para a coluna 3, até o preenchimento do registrador. Os elementos mais a direita em cada linha do registrador são enviados a saída, para cada palavra-código recebida, fazendo com que as $D - 1$ primeiras palavras-código na saída do *de-interleaver* sejam sempre zeros (representados pelo x na Figura 4.7), caracterizando o *delay* ou atraso do *interleaver*. A partir do preenchimento do registrador é que a informação desembaralhada é então enviada pra saída.

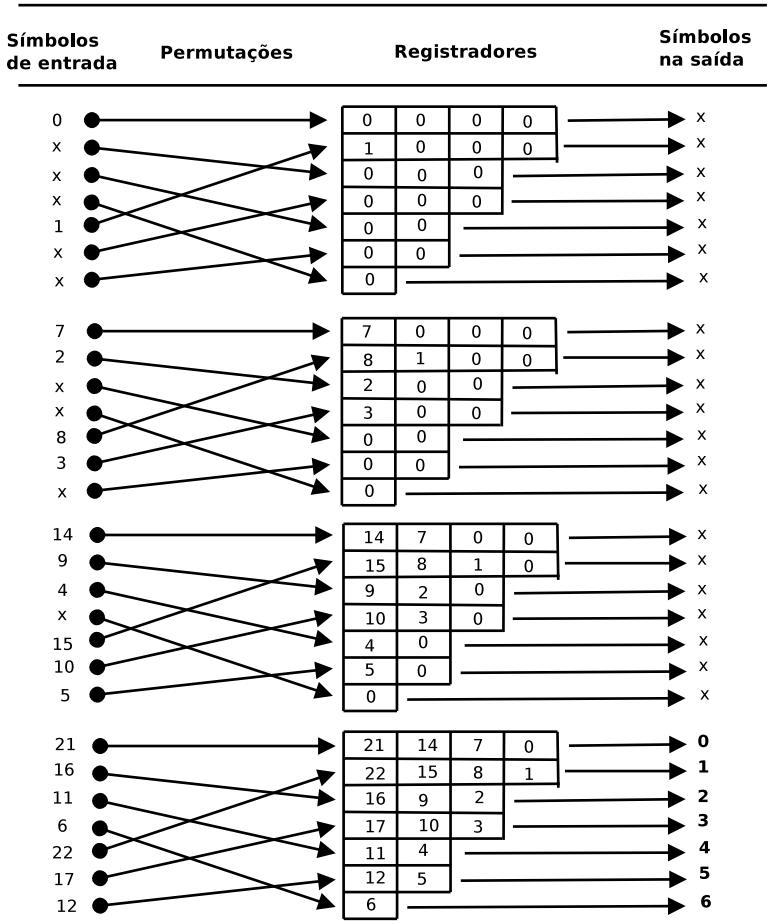


Figura 4.7: Funcionamento do *de-interleaver* com $N = 7$ e $D = 4$, descrito em [37] .

Capítulo 5

Implementação em Software da Codificação de Canal ADSL utilizando Ptolemy

O software usado para implementação da codificação de canal do Software Modem ADSL foi o Ptolemy II. A escolha do Ptolemy é baseada na possibilidade e facilidade de modelar e simular sistemas complexos como o caso do Software Modem ADSL.

5.1 Ptolemy II

O Projeto Ptolemy [4] é um grupo informal de pesquisadores da Universidade da Califórnia em Berkeley que estuda a modelagem heterogênea, design e simulação de sistemas, em especial, sistemas embarcados. O foco é em sistemas embarcados que operam com diferentes tecnologias incluindo, processamento de sinais, redes, interfaces de usuários e *feedback control*, por exemplo. A vantagem do projeto está na utilização de uma metodologia baseada em componentes usando modelos de computação que regem a interação entre eles.

O Ptolemy II é o atual software usado pelo projeto. É um *framework open-source* construído em Java que permite a simulação de diversos sistemas. No Ptolemy II, os chamados *modelos* são construídos graficamente através de uma interface chamada Vergil, como mostrado na Figura 5.1. Os *modelos* caracterizam-se por uma interconexão de *atores* (componentes do software) que são executados e se comunicam através de portas.

Na Figura 5.1, cada bloco é um *ator* que representa uma função ou um conjunto de

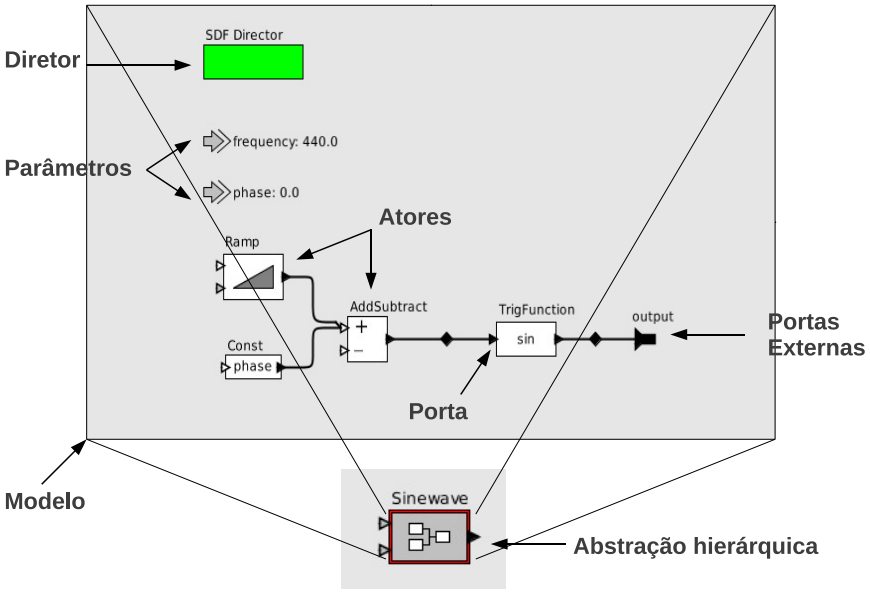


Figura 5.1: Um exemplo de *modelo* construído dentro do Vergil, a interface gráfica do Ptolemy II.

funções que recebe informações de uma porta de entrada e produz um resultado em uma porta de saída. As informações são passadas pelas portas através de *tokens*. Além disso, a interface também possibilita a definição de *parâmetros* que são usados para configurar a operação de um *ator*.

Os *modelos* podem definir uma interface externa, chamada de *abstração hierárquica*. Essa interface consiste de portas e parâmetros externos que podem ser conectados a outras portas externas do modelo ou a portas de *atores* que compõem o modelo.

Os *atores* são executados através de um *Diretor* que determina como esses componentes devem ser agendados para execução. Essas execuções são chamadas de *firing*. Os *Diretores* implementam os modelos de computação baseados em *domínios*. Alguns dos *domínios* usados no Ptolemy são: evento discreto (DE), tempo contínuo (CT), fluxo de dados síncrono (SDF), máquinas de estado finito (FSM), Giotto, Rendezvous, processos de comunicação sequencial (CSP), redes de processos (PN), fluxo de dados dinâmicos (DDF), entre outros [38].

O domínio usado na simulação do Software Modem ADSL é o domínio SDF. O domínio SDF é útil para a modelagem de sistemas de fluxos de dados simples, como o caso dos sistemas de processamento de sinais. No domínio SDF, a ordem de execução dos atores é determinada antes da execução. O diretor SDF define quantos iterações serão rodadas, ou seja, quantos *firing* serão executados nos atores. Cada ator, a cada iteração consome *tokens* na entrada,

realiza as suas operações e devolve *tokens* na saída para cada iteração. Os tipos de *tokens* definem os tipos das portas, ou seja, cada *ator* pode ter suas portas definidas como inteiros, bytes, números complexos, *strings*, *boolean*, *double* e etc, ou vetores de inteiros, bytes, *boolean* e etc, e também como matrizes.

O Ptolemy II vem como uma biblioteca de *atores* já prontos que podem ser usados ou modificados, ou atores podem ser criados e implementados em Java e instanciados para o Ptolemy, bastando que para isso os atores criados sigam e implementem os métodos necessários nos *atores* do Ptolemy. Ao todo são oito métodos: *preinitialize()*, *initialize()*, *prefire()*, *fire()*, *postfire()*, e *wrapup()*. O método *initialize()* é o responsável pela inicialização das variáveis utilizadas pelo ator. Já o método *fire()* é o método no qual ocorre toda a ação do ator. Neste método são lidos os *tokens* na entrada e é feita a operação do ator, assim como o envio do resultado na forma de *tokens* para a porta de saída. Os outros métodos são apenas métodos auxiliares que executam funções definidas pelo Ptolemy [39].

5.2 Descrição da Implementação dos Blocos de Codificação de Canal

Os blocos responsáveis pela codificação de canal e modulação do Software Modem ADSL usados na fase de *Showtime* do modem ADSL foram implementados na plataforma Ptolemy II seguindo as especificações do Padrão ADSL ITU-T G.992.1 [3]. A fase de *Showtime* de um modem ADSL é a fase onde os modens começam a transmitir e receber dados, após passarem pelas fases de inicialização (*Training*, *Channel Analysis*, *Exchange*) [3]. Os blocos do transmissor e receptor implementados no software modem são mostrados no diagrama de blocos da Figura 5.2. Comparando com o diagrama de blocos da Figura 4.1, os blocos TEQ (*Time-Domain Equalizer*) e FEQ (*Frequency-Domain Equalizer*) não foram incluídos nas simulações pois ainda estão em processo de implementação e testes. Apesar dos blocos de modulação não fazerem parte do escopo desse trabalho, se faz necessária a explicação da sua implementação, pois esses blocos serão usados nas simulações da fase de *showtime* usando transmissão de vídeo que será explicada no próximo capítulo.

Cada um dos blocos responsáveis pela codificação e modulação foram criados e implementados em Java e posteriormente instanciados dentro do Ptolemy, seguindo a implementação dos métodos responsáveis pelas operações no Ptolemy. Por exemplo, na implementação do bloco *scrambler*, primeiramente a classe *Scrambler* é criada composta por métodos responsáveis por realizar o embaralhamento dos bits, depois a classe *ScramblerActor*

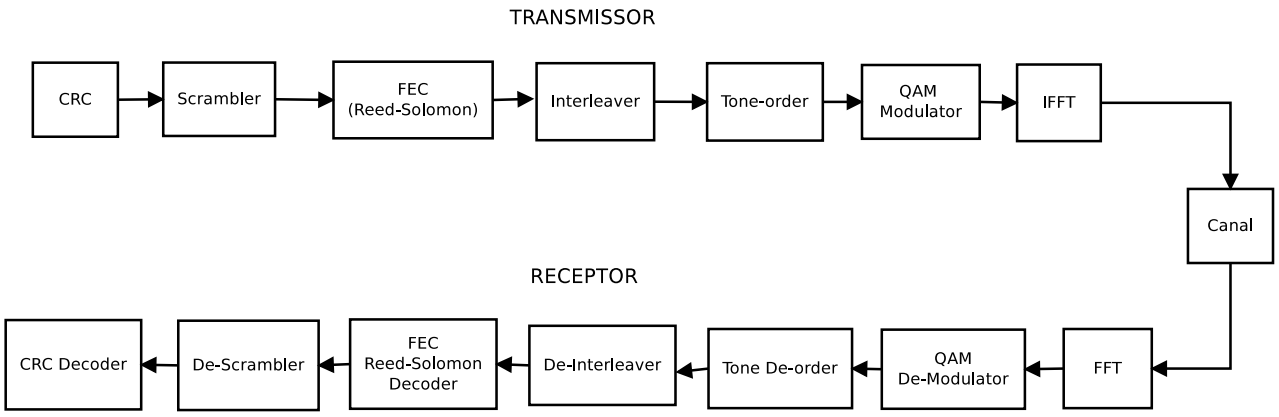


Figura 5.2: Diagrama de blocos do transmissor e receptor do Software modem ADSL implementados no Ptolemy.

é criada, com métodos usados pelo Ptolemy para executar as ações (ex: *fire()*, *initialize()*). Além disso, a classe *ScramblerActor* agrega a classe *Scrambler*, podendo então realizar as operações do *scrambler*. A Figura 5.3 mostra o diagrama de classes do *ScramblerActor*. Já a Figura 5.4 mostra o ator *ScramblerActor* no Ptolemy.

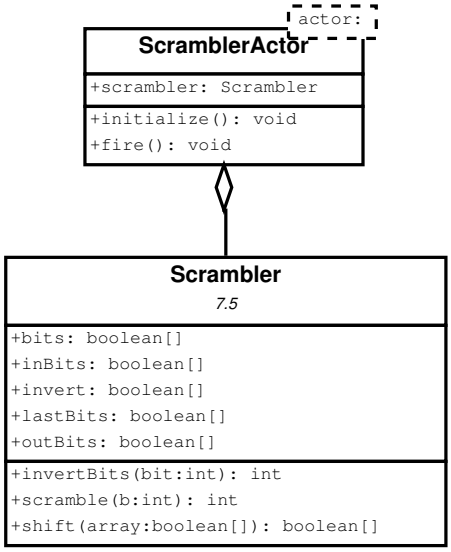


Figura 5.3: Diagrama de classes da classe *ScramblerActor*.

Quase todos os blocos da Figura 5.2 foram implementados dessa forma, com exceção das classes *FFTActor* e *IFFTActor* que foram implementadas usando a biblioteca já pronta do Ptolemy, facilitando a implementação.

Além disso, alguns blocos precisam que os seus parâmetros sejam configurados. Por

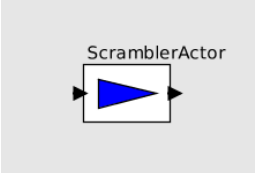


Figura 5.4: *ScramblerActor*.

exemplo, no caso do *ReedSolomonActor*, os parâmetros relacionados a codificação, como tamanho da palavra-código (*codewordLength*) e o número de bytes de paridade (*parityBytes*) podem ser definidos, possibilitando liberdade no uso do ator. Dessa forma, pode-se definir qualquer tamanho de palavra-código, assim como a paridade que será adicionada. A Figura 5.5 mostra como são configurados esse parâmetros. Os outros blocos que dependem de parâmetros são configurados igualmente. O *CRCActor* necessita dos seguintes parâmetros: *CRCCode*, que define o polinômio gerador usado, como o definido para o ADSL na Equação 4.3, o tamanho do bloco, e o número de *frames* (68, representando o tamanho de um *superframe*, ou seja, o CRC será colocado após 68 *frames*). E o *ConvInterleaverActor* necessita dos seguintes parâmetros: tamanho do bloco e o *interleaver depth*. No caso do *Scrambler* nenhum parâmetro necessita ser configurado, pois o polinômio usado no ADSL já é implementado na classe *Scrambler*. Os mesmos parâmetros são configurados nos blocos do receptor.

É importante ressaltar a dificuldade na implementação dos blocos do receptor. O padrão do ADSL [3] apenas define o modelo de funcionamento dos blocos do transmissor, pois é necessário que os sinais e mensagens transmitidos possam ser processados por equipamentos de diferentes fabricantes. Já a implementação do receptor fica a cargo de cada fabricante e não é definida no padrão.

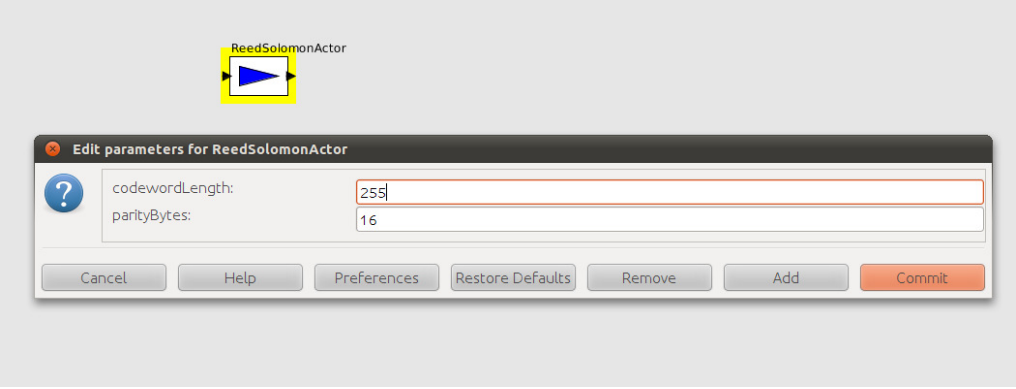


Figura 5.5: Configuração de parâmetros do bloco *ReedSolomonActor*.

A Figura 5.6 mostra um modelo no Ptolemy para fase de *Showtime* contendo todos os blocos do transmissor e receptor do Software Modem ADSL, considerando apenas o *interleaver path*. Esse modelo consiste em um exemplo simples no qual ainda não foi considerada a presença de um canal responsável por alterar o sinal transmitido e nem a presença de ruído. No ramo superior da Figura 5.6 estão os blocos do transmissor e no ramo inferior os blocos do receptor. No Ptolemy é possível plotar o sinal através de um *SequencePlotter*, assim como, por exemplo, verificar se o CRC detectou algum erro, usando um *display* que mostra uma mensagem informando se ocorreram erros ou não.

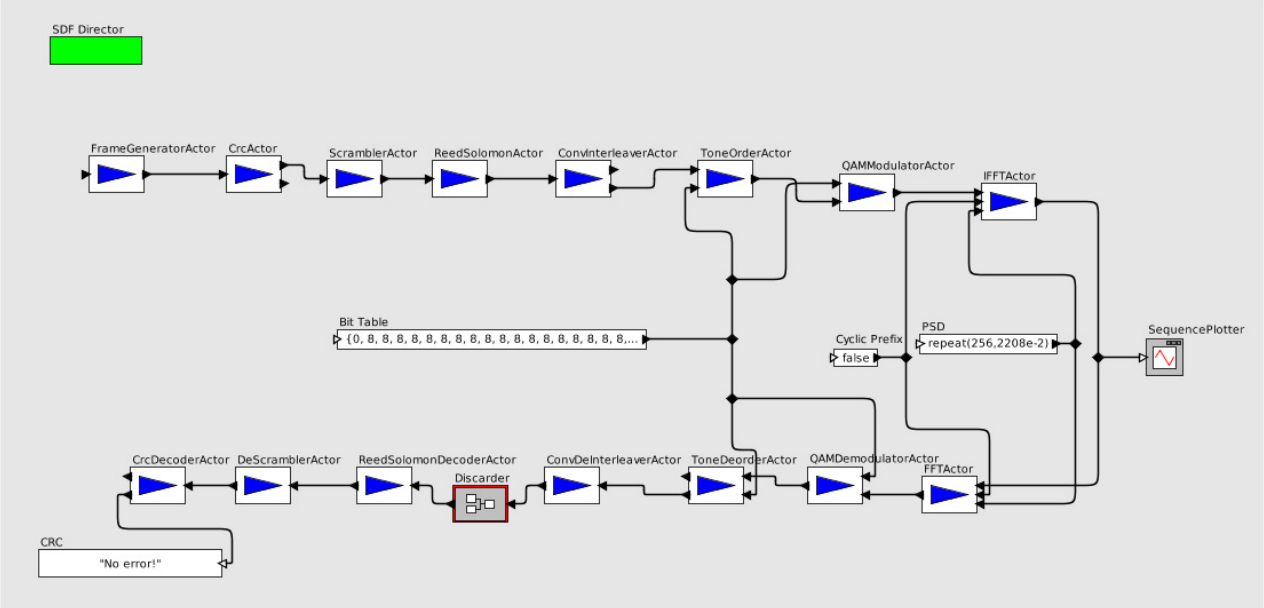


Figura 5.6: Modelo no Ptolemy dos blocos do transmissor e receptor do Software modem ADSL.

5.3 Validação com o TraceSpan

Para validar o Software modem ADSL aqui implementado, ou seja, garantir que as suas especificações e o seu funcionamento estejam de acordo com o sugerido no padrão do ADSL [3] e de acordo com uma transmissão real DSL, foi usado o analisador TraceSpan (http://www.tracespan.com/), que é uma ferramenta que fornece uma análise precisa das informações da camada física.

O analisador TraceSpan verifica a interoperabilidade dos produtos de diferentes fornecedores e compara o desempenho de qualquer combinação de ATU-C e modens ATU-R.

Ele fornece uma análise completa e precisa, enfocando as informações da camada física. E uma vez que é uma tecnologia não invasiva, não afeta a transferência de dados entre os transceptores. Com o TraceSpan, é possível realizar uma análise detalhada dos dados transmitidos entre ATU-C e ATU-R, em qualquer formato, desde de amostras, constelações, mensagens até bits. Além de permitir a visualização de dados tanto para *downstream* quanto para *upstream* [40].

Como essa análise pode fornecer informações em padrões de bits, é possível então comparar uma transmissão real, gravada e analisada pelo TraceSpan com o gerado pelo Software modem ADSL.

A Figura 5.7 mostra como o analisador TraceSpan pode ser configurado entre o CO e o lado do usuário, sem interferir na transmissão.

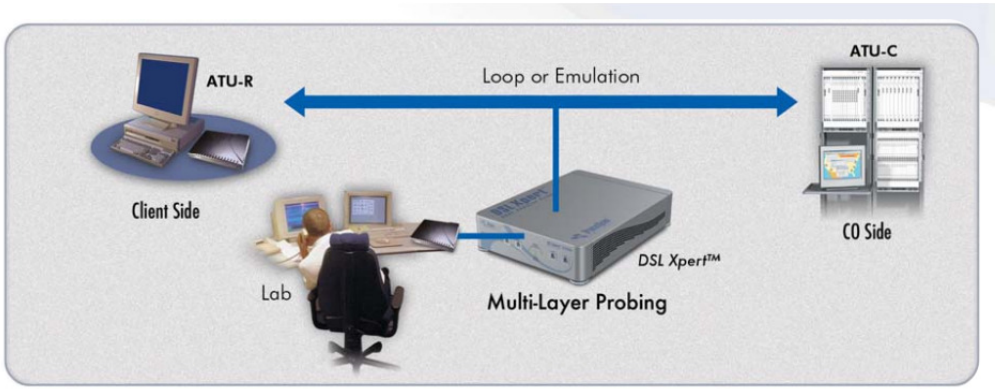


Figura 5.7: Configuração do analizador TraceSpan [40].

A validação com o TraceSpan é realizada *off-line*. Uma transmissão real é gravada pelo TraceSpan, como mostrado na Figura 5.7. Com os dados exportados pelo TraceSpan referentes a essa transmissão, consistindo em arquivos binários, é possível testar cada bloco do Software modem separadamente, comparando os dados obtidos pelo TraceSpan com os dados obtidos pelo Software modem, considerando a mesma entrada. No entanto, com o uso do TraceSpan é possível apenas validar os blocos do receptor do Software modem. E os blocos aqui validados serão apenas os blocos referentes a codificação de canal do receptor de um modem ADSL: *De-interleaver*, *ReedSolomon decoder*, *de-scrambler*, *CRC decoder*.

O TraceSpan pode exportar dados referentes a fase de *showtime* em 3 fases distintas, como mostrado na Figura 5.8 [40].

Além disso, o TraceSpan pode exportar dados referentes ao sinal em quatro fases distintas, conforme ilustrado na Figura 5.9.

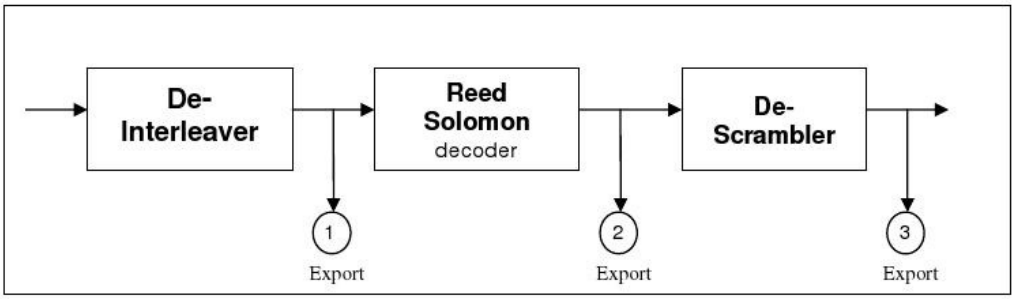


Figura 5.8: Os 3 pontos de dados exportados pelo TraceSpan na fase de *Showtime* [40].

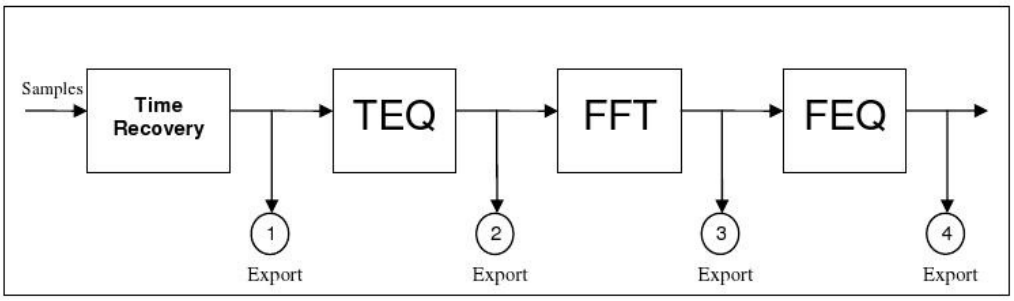


Figura 5.9: Os 4 pontos de visualização do sinal do TraceSpan [40].

Os 3 pontos de dados exportados pelo TraceSpan na fase de *showtime* são:

1. Palavra-código FEC com redundância (antes da decodificação do Reed-Solomon e depois do *De-interleaver*)
2. Palavra-código FEC corrigida sem redundância (depois da decodificação do Reed-Solomon)
3. Dados após do *De-scrambler*.

Os dados são exportados tanto para o *downstream* quanto para o *upstream*. Cada dado exportado é um arquivo binário denominado:

1. FEC LP1 R.bin e FEC LP1 C.bin (representando o ATU-R e o ATU-C)
2. COR FEC LP1 R.bin e COR FEC LP1 C.bin
3. SCRAM LP1 R.bin e SCRAM LP1 C.bin

Com os dados exportados pelo TraceSpan para a fase de *showtime*, mostrados na Figura 5.8, é possível apenas validar os blocos que possuem dados de entrada e saída, ou seja, os blocos *ReedSolomon decoder* e *De-scrambler*. O bloco *De-interleaver* será validado com as informações do sinal conforme mostrado na Figura 5.9, e o bloco *CRC decoder* será validado usando outra estratégia, que será mostrada posteriormente.

Para captura dos dados, um setup foi montado em laboratório, consistindo de DSLAM EDN312xp, modem Speedstream 4200 e um cabo de 2000m, com o modo "trellis" desabilitado por não constar nas implementações do Software modem. O tempo de captura foi de 43 s.

A Figura 5.10 mostra o modelo no Ptolemy usado para a validação dos blocos *ReedSolomonDecoderActor* e *DeScramblerActor*.

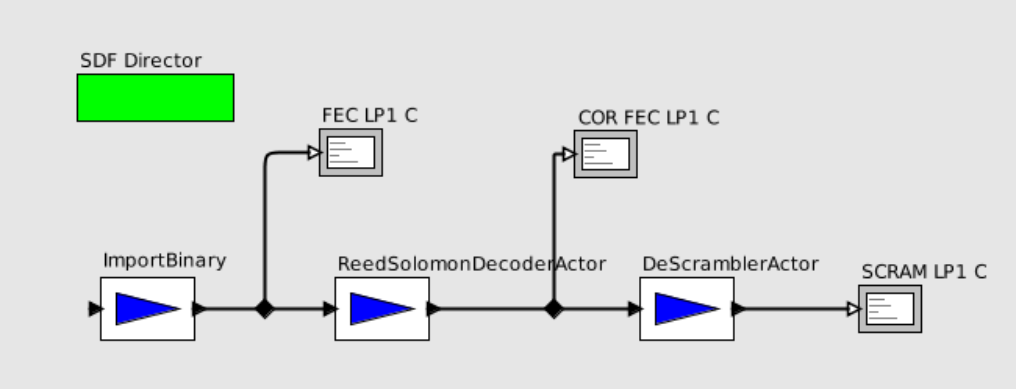


Figura 5.10: Modelo no Ptolemy para validação dos blocos *ReedSolomonDecoderActor* e *DeScramblerActor*.

O bloco *ImportBinary* é responsável por importar o arquivo FEC LP1 C.bin que é usado como entrada do bloco *ReedSolomonDecoderActor*.

Validação dos blocos *Reed-Solomon decoder* e *De-Scrambler*

Para a validação do bloco *ReedSolomon decoder*, o arquivo binário FEC LP1 C.bin é dividido em *frames* e cada *frame* é decodificado usando o bloco *ReedSolomonDecoderActor* e comparado com os *frames* decodificados do arquivo binário COR FEC LP1 C.bin. Nos arquivos do TraceSpan observou-se que o código Reed-Solomon utilizado para decodificar cada *frame* foi o *shortened* RS(143, 129) (ver Seção 3.2.4), com o tamanho do *frame* N=143, K= 129 bytes de informação e R = 14 bytes de redundância ou paridade.

Como um exemplo, considere o primeiro *frame* do arquivo binário FEC LP1 C.bin contendo 143 bytes (descartando os 128 bytes de cabeçalho):

```
frame 1 = 0 0 0 0 128 74 86 60 42 140 224 195 32 27 6 24 113 18 57 0 28 109 173 231 222 149 213 5 36 69 42 62
51 126 192 114 173 56 199 126 74 82 220 102 88 245 82 121 49 26 47 38 103 217 15 214 211 223 238 112 48 80 29
188 185 143 73 147 144 152 214 247 203 233 9 255 107 192 232 140 123 212 187 255 150 176 239 67 172 50 90 150
113 116 13 233 143 116 157 195 153 150 96 242 45 29 170 226 166 223 106 231 160 89 129 199 216 175 221 142 124
136 104 162 94 5 147 2 118 23 116 207 78 254 151 246 13 83 249 217 186 157 235
```

Os últimos 14 bytes são a redundância:

```
R = 23 116 207 78 254 151 246 13 83 249 217 186 157 235
```

O resultado da decodificação é:

```
frame decoded 1 = 0 0 0 0 128 74 86 60 42 140 224 195 32 27 6 24 113 18 57 0 28 109 173 231 222 149 213 5 36 69
42 62 51 126 192 114 173 56 199 126 74 82 220 102 88 245 82 121 49 26 47 38 103 217 15 214 211 223 238 112 48 80
29 188 185 143 73 147 144 152 214 247 203 233 9 255 107 192 232 140 123 212 187 255 150 176 239 67 172 50 90 150
113 116 13 233 143 116 157 195 153 150 96 242 45 29 170 226 166 223 106 231 160 89 129 199 216 175 221 142 124 136
104 162 94 5 147 2 118
```

Este resultado corresponde ao mesmo *frame* 1, apenas sem a redundância. Os resultados foram idênticos aos resultados da TraceSpan para todos os *frames* comparados dessa transmissão. No caso dessa simulação não houveram erros pois o canal era perfeito e nenhum ruído foi adicionado. No entanto, apesar do Reed-Solomon não necessitar realizar a correção, para que o bloco funcione é necessário que ele decodifique corretamente os *frames*.

Para a validação do bloco *DeScramblerActor*, todo o arquivo binário COR FEC C.bin LP1 é desembaralhado bit a bit e comparado com o arquivo SCRAM LP1 C.bin do TraceSpan. Então, os bits são embaralhados através do armazenamento em um *buffer* dentro do bloco e são mandados para a saída na forma de *frames*. Tomando como entrada o *frame decoded* 1, contendo 129 bytes, o *frame* desembaralhado correspondente ao *frame* do arquivo SCRAM LP1 C.bin é:

```
frame deScrambled 1 = 0 0 0 0 128 74 86 86 86 86 86 230 228 228 228 228 100 113 113 113 113 113 221 221 221 221
221 189 184 184 184 184 184 147 147 147 147 147 203 202 202 202 202 10 0 0 0 0 0 86 86 86 86 86 0 0 0 128 74 230
228 228 228 228 100 113 113 113 113 113 221 221 221 221 221 189 184 184 184 184 184 147 147 147 147 147 203 202
202 202 202 10 0 0 0 0 0 86 86 86 86 86 230 228 228 228 228 0 0 0 128 74 100 113 113 113 113 113 221 221 221 221
221 189 184 184 184 184 184
```

A validação do bloco *ConvDeInterleaverActor* foi realizada com auxilio das informações de sinal exportado na Figura 5.8. A Figura 5.11 mostra a integração dos blocos do receptor assim como os pontos onde os dados podem ser exportados.

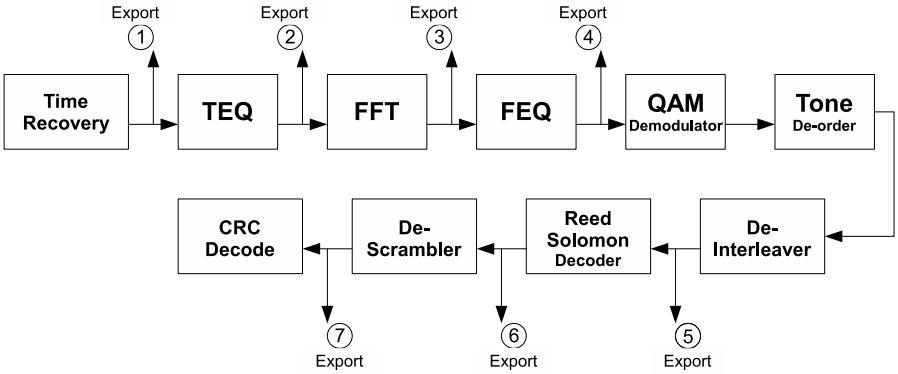


Figura 5.11: Diagrama de blocos do receptor mostrando em que pontos os dados podem ser exportados.

Validação do bloco *De-interleaver*

É possível notar pela Figura 5.11 que não é possível validar o *De-interleaver* por não ser possível exportar os dados na entrada do bloco, assim a estratégica aqui adotada é de validar 3 blocos juntos: *QAM de-modulator*,*Tone De-order* e *De-interleaver*. A validação então consiste em integrar os 3 blocos alimentando a entrada do *QAM de-modulator* do Software modem com o sinal exportado pelo TraceSpan depois do FEQ (ponto 4 da Figura 5.11) e comparando a saída do *de-interleaver* do Software modem com os dados binários exportados pelo TraceSpan no ponto 5.

Foram testados vários *superframes* e os dados na saída do bloco *de-interleaver* foram idênticos aos dados binários do TraceSpan - excluindo os primeiros 64 *frames* que correspondem ao atraso do *interleaver* e que são ignorados e descartados pelo TraceSpan.

A Figura 5.12 representa o modelo no Ptolemy para validação do bloco *ConvDeInterleaverActor* com os 3 blocos ligados. O bloco *TracespanReaderFreqDomain* é responsável por importar o sinal do TraceSpan. E a constante *Bit Table* é configurada com o *bitloading* que também é salvo durante a transmissão pelo TraceSpan.

Validação do bloco *Crc decoder*

Com o Relatório de Análise do TraceSpan (um relatório gerado com todas as informações sobre a gravação, os parâmetros de inicialização, protocolos de *showtime* e gráficos

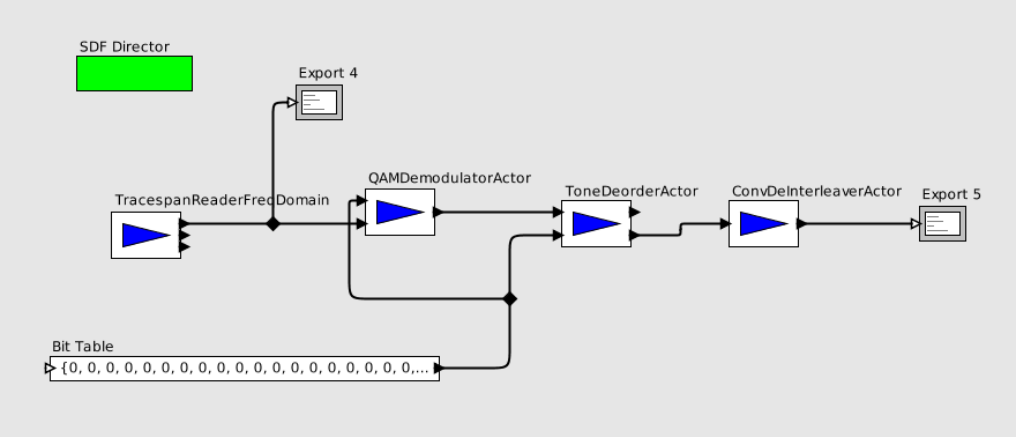


Figura 5.12: Modelo no Ptolemy para validação do bloco *ConvDeInterleaverActor*.

de *Channel Analysis*), é possível visualizar um *Showtime/UpperLayer Overview*, um resumo constando algumas informações referentes a fase de *showtime*.

Nesse resumo, uma tabela mostra os *superframes* de *downstream/upstream* transmitidos durante a fase de *showtime* de um modem ADSL. Para cada *superframe* (composto de 68 *frames*) do *interleaved buffer* (o *fast buffer*, sem *interleaver*, não foi considerado nos testes) uma tabela com todos os *sync bytes* de cada *frame* é mostrada.

Lembrando que no *interleaver buffer* cada *frame* é composto por um *sinc byte* como o primeiro byte do *frame*, onde no *frame* 0, o *sync byte* carrega o CRC do *superframe*, e em todos os outros *frames* (1-67), este byte é usado para controle de sincronização ou para controle de *overhead* (AOC).

A Tabela 5.1 mostra os *sync bytes* de 10 *frames* de um *superframe*.

Sabendo o valor do CRC (em hexadecimal), comparou-se com o CRC calculado no transmissor do Software modem e os resultados foram os mesmos. Para validação do *CRC decoder*, o valor do CRC recalculado deve resultar em zero, garantindo que não ocorreram erros na transmissão. Isso só pode acontecer se o valor do CRC tiver sido calculado corretamente. Usando o *CRCDecoderActor*, os *frames* do arquivo binário SCRAM LP1 C.bin são armazenados no bloco e quando um *superframe* é completado, recalcula-se o CRC. Se o CRC é igual a zero significa que nenhum erro foi encontrado e a mensagem "No Error! "é exibida. Para todos os *superframes* exportados, o *CRCDecoderActor* conseguiu decodificar corretamente e o CRC recalculado resultou em zero.

A Figura 5.13 mostra o modelo no Ptolemy para a validação completa dos blocos de codificação da fase de *showtime* do receptor do Software modem ADSL (referente a

Tabela 5.1: *Sync Bytes* de um *superframe*.

Frame ID	Dados de *Overhead*	Tipo de *Overhead* esperado
0	0x9C	CRC
1	0xFF	IB
2	0x0C	SYNC
3	0x0C	SYNC
4	0x00	AOC
5	0x00	AOC
6	0x0C	SYNC
7	0x0C	SYNC
8	0x00	AOC
9	0x00	AOC

Figura 5.11), junto com a validação adicional dos blocos de modulação. Os blocos foram primeiro validados individualmente e depois validados conjuntamente.

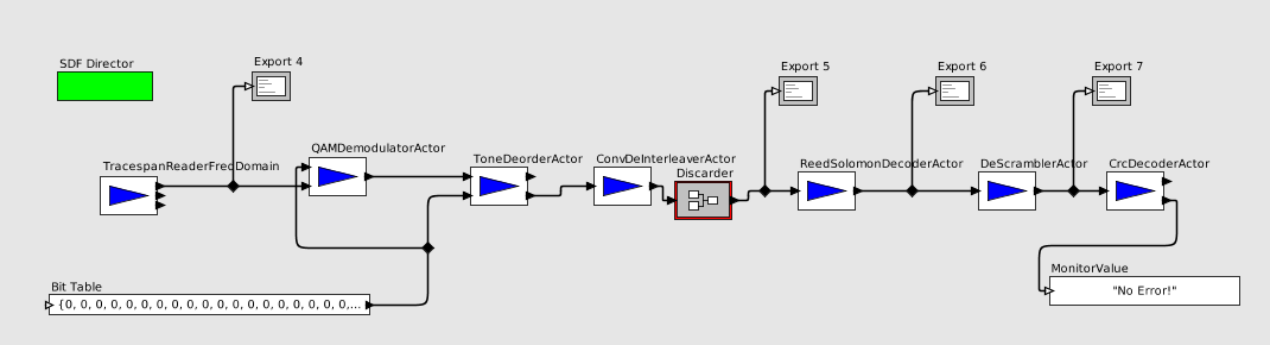


Figura 5.13: Modelo no Ptolemy para validação dos blocos da codificação de canal do receptor do Software modem ADSL.

Os blocos responsáveis pela codificação de canal do transmissor do Software modem ADSL (*CrcActor*, *ScramblerActor*, *ReedSolomonActor* e *ConvInterleaverActor*) foram validados baseados na validação dos blocos do receptor. Cada bloco foi testado separadamente usando os blocos do receptor para garantir que estivessem dentro dos padrões do ADSL e de acordo com uma transmissão real DSL.

É importante ressaltar a dificuldade na validação dos blocos, especialmente dos blocos

do receptor, devido a falta de informação sobre o funcionamento e as especificações do receptor, visto que o padrão do ADSL [3] apenas define essas especificações para o transmissor. Cada bloco precisou ser cuidadosamente estudado e implementado de maneira correta o que demandou um certo tempo e exaustivos testes.

5.4 Implementação do Modelo de Ruído Impulsivo

O modelo de geração de ruído impulsivo descrito na Seção 2.4.3 foi implementando no Ptolemy seguindo a descrição de implementação sugerida em [16].

O modelo de ruído impulsivo no Ptolemy foi criado a partir de duas máquinas de estados finitas, usando o domínio FSM (*Finite State Machine*) do Ptolemy. Esse domínio permite a criação de máquinas de n estados. O modelo de ruído impulsivo implementado no Ptolemy é mostrado na Figura 5.14.

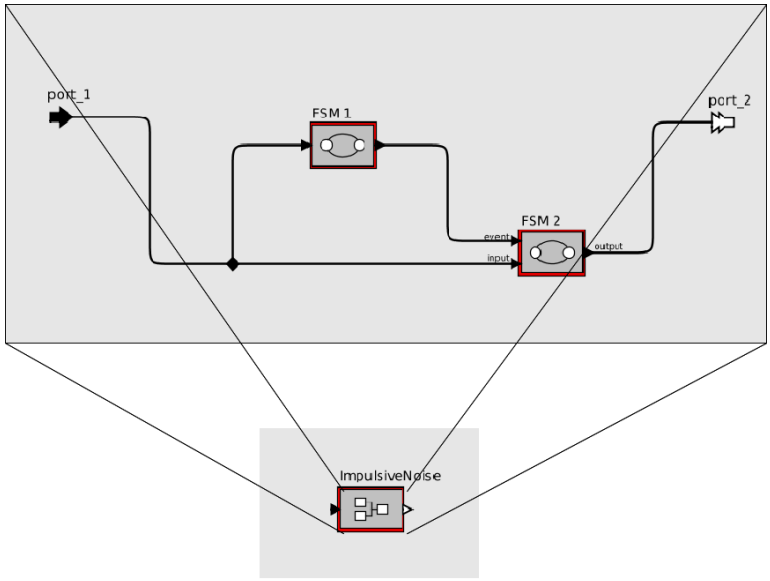


Figura 5.14: Modelo de ruído impulsivo no Ptolemy.

A máquina de estados denominada FSM 1, mostrada na Figura 5.15 é uma máquina de 2 estados, que alterna entre um estado com ruído e um estado sem ruído impulsivo. Os impulsos são gerados no tempo e depois convertidos em amostras. A mudança de um estado para o outro é realizada através da contagem de amostras, utilizando a frequência de amostragem do ADSL.

O estado denominado *gapTime* é responsável por gerar o intervalo entre ruídos. A classe *GapTimeCounter* implementada dentro do estado *gapTime* calcula o intervalo entre ruídos

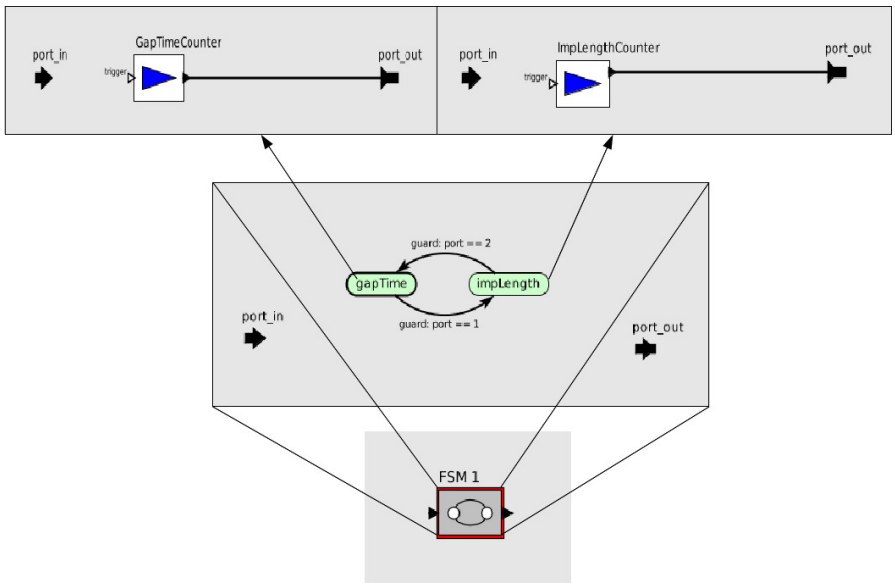


Figura 5.15: Máquina de estados FSM 1.

em segundos, usando a distribuição de probabilidade descrita na Seção 2.4.3 e multiplicando pela frequência de amostragem de aproximadamente 2,2 MHz do ADSL configurada como um parâmetro no Ptolemy retorna o valor em número de amostras. Essa classe então "conta" as amostras e quando a contagem termina, a mudança de estado é realizada.

O outro estado denominado *impLength* é responsável por gerar a duração do ruído. A classe *ImpulseLengthCounter* implementada dentro do estado *impLength* calcula a duração do ruído impulsivo em segundos e multiplicando também pela frequência de amostragem de aproximadamente 2,2 MHz do ADSL, retorna o mesmo valor em número de amostras. Essa classe então "conta" as amostras que equivalem a duração do ruído e ao terminar muda de estados.

Ambos os estados, ao terminarem as suas contagens de amostras, mandam para a porta de saída (*port out*) uma indicação de que o evento mudou, ou seja, de que um estado que por exemplo era caracterizado pela ausência de ruído agora é um estado com ruído impulsivo. Essa indicação de evento é mandada para a outra máquina de estados denominada FSM 2.

A máquina de estados FSM 2, mostrada na Figura 5.16, é também uma máquina de 2 estados, que também alterna entre um estado sem ruído impulsivo chamado de *clean* e um estado com ruído impulsivo chamado de *noisy*. No estado *clean*, o que estiver na porta de entrada (*input*) é mandado diretamente para a porta de saída (*output*), representando o intervalo sem ruído. No estado *noisy*, para cada amostra proveniente da porta de entrada, a classe *ImpulseAmplitudeActor* responsável por gerar a amplitude do ruído, gera um valor de

amplitude que é somado com o valor da amostra, resultando assim em um ruído impulsivo com amplitudes e durações definidas. A mudança de estado nesse caso é realizada através da porta *event*, ou seja, o estado é mudado assim que a porta *event* receber uma indicação proveniente da máquina de estados FSM 1 pedindo a mudança de estado após a contagem de amostras.

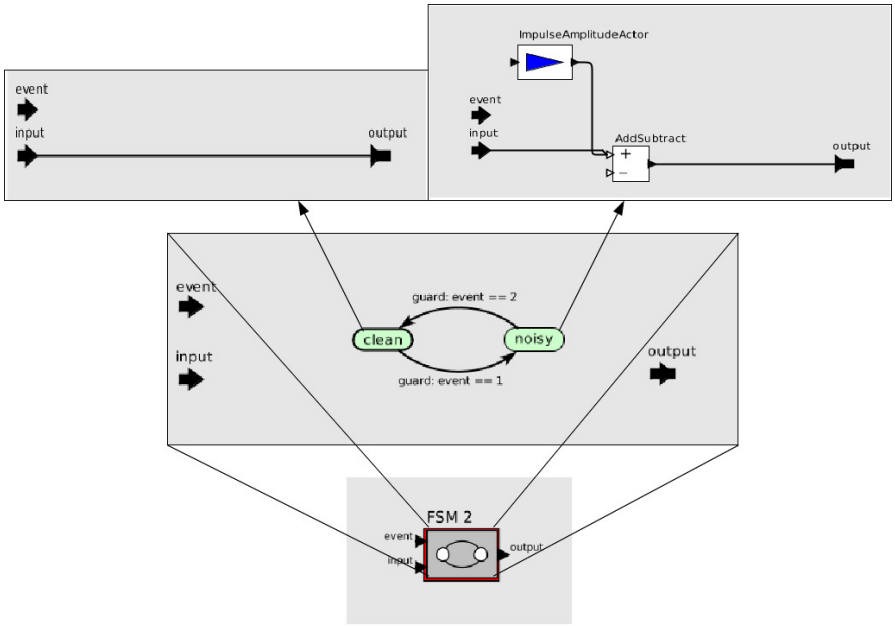


Figura 5.16: Máquina de estados FSM 2

Capítulo 6

Resultados

6.1 Introdução

Este capítulo apresenta os resultados das simulações realizadas no Software modem ADSL. Para as simulações em *showtime*, um cenário com transmissão de vídeo e usando um canal com ruído impulsivo foi considerado. A transmissão de vídeo foi usada nesse trabalho por ser uma das mais importantes aplicações para o usuário final. O ADSL, por exemplo, foi originalmente desenvolvido para aplicações de transmissão de serviços de vídeo. Além disso, os serviços de transmissão de vídeo pela internet (IPTV, vídeo conferência, video-on-demand) estão cada vez mais em evidência e exigindo um certo padrão de qualidade de serviço.

O objetivo aqui é mostrar uma das possíveis aplicações da codificação de canal do Software modem ADSL.

6.2 Descrição da Simulação

Para as simulações em *showtime*, um cenário com transmissão de vídeo e usando um canal com ruído impulsivo foi considerado e será descrito a seguir. A Figura 6.1 mostra o modelo no Ptolemy para a simulação de transmissão de vídeo. Na parte superior da figura estão os blocos do transmissor, seguido pelo modelo de ruído impulsivo e na parte inferior estão os blocos do receptor. As simulações consideraram apenas a direção *downstream* (central para usuário), por possuir uma faixa de frequência maior que o *upstream*, sendo mais suscetível a ruídos e por que a direção *downstream* é mais importante considerando a perspectiva do usuário.

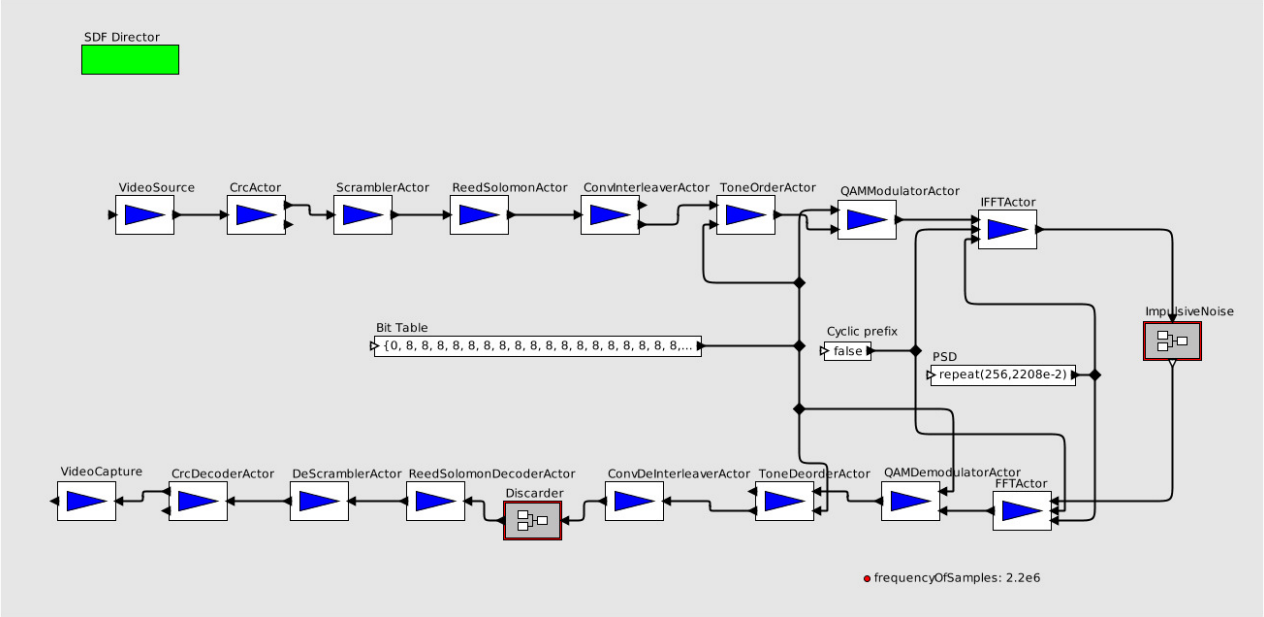


Figura 6.1: Modelo no Ptolemy para transmissão de vídeo na presença de ruído impulsivo.

6.2.1 Blocos do Transmissor

Os blocos do transmissor são constituídos por:

- **VideoSource**: Esse bloco é responsável em ler o *bit stream* do vídeo e gerar os *frames* necessário para a transmissão. O número de *frames* lidos, correspondendo ao tamanho do vídeo, assim como o tamanho dos *frames* também são especificados nesse bloco.
- Blocos de codificação: **CRCActor**, **ScramblerActor**, **ReedSolomonActor**, **ConvInterleaverActor**;
- **ToneOrderActor**, **QAMModulatorActor**, **IFFTActor**.

6.2.2 Blocos do Receptor

Os blocos do receptor são formados por:

- **FFTActor**, **QAMDemodulatorActor**, **ToneDeorderActor**
- Blocos de decodificação: **ConvDeInterleaverActor** (com um bloco denominado *Discarder* que descarta os primeiros (*interleaver depth* - 1) *frames*, que correspondem aos primeiros frames resultantes do *ConvDeInterleaverActor*, provenientes dos frames

contendo zeros usados para preencher o registrador, como explicado na Seção 4.6; **ReedSolomonDecoderActor**; **DeScramblerActor** e **CRCDecoderActor**.

- **VideoCapture**: Esse bloco é responsável por recuperar o vídeo transmitido, salvando os *frames* recebidos em um outro vídeo que será utilizado para posterior comparação com o vídeo original.

Os parâmetros utilizados em ambos os blocos do transmissor e receptor foram os recomendados em [3] e listados na Tabela 6.1. Nas simulações, o número de bytes de paridade foi configurado para a maior capacidade de correção permitida no ADSL (16 bytes por palavra-código), ou seja o código RS utilizado foi o RS(255, 239). O valor do *interleaver depth* foi variado para permitir uma comparação da qualidade do vídeo na presença de ruído impulsivo e para verificar como ele modifica a performance da correção de erros. O canal utilizado nas simulações considerou apenas a presença de ruído impulsivo, portanto, o *bit loading* utilizado foi um *bit loading* com configuração *flat* com 8 bits por subcanal (representado pela constante *Bit Table* na Figura 6.1). A PSD (*Power Spectral Density*) utilizada em cada tom foi calculada como em [41], ou seja, o valor utilizado foi de -40 dbm/Hz.

Tabela 6.1: Parâmetros usados pelos blocos do transmissor e receptor.

Parâmetros	Valores
Tamanho do frame de informação	239 bytes
Tamanho do superframe	68 frames
Tamanho da palavra-código (com paridade)	255 bytes
Bytes de paridade	16 bytes
Número de símbolos DMT por palavra-código	1
Interleaver depth	2, 4, 8, 16, 32, 64
Número de tons	256
Número de pontos da FFT/IFFT	512
Prefixo Cíclico	32 amostras

Os parâmetro utilizados na simulação do ruído impulsivo foram os parâmetros referentes as medições da BT (British Telecom), citados na Seção 2.4.3.

Nas simulações, vídeo *streams* no formato MPEG-4 foram transmitidas pelo cenário de *showtime* do Software modem ADSL acima descrito e o impacto do ruído impulsivo na qualidade do vídeo foi avaliado. O formato MPEG-4 [42] foi escolhido pois caracteriza-se por

ser um formato relativamente novo em comparação ao MPEG-2, apresentando a vantagem de atingir maiores taxas de compressão mantendo a qualidade, sendo portanto mais adequados para o uso de aplicações de transmissão de vídeos pela internet.

As simulações foram realizadas utilizando-se 2 tipos de MPEG-4 vídeo *stream* (não contendo áudio):

- "news.mp4" - codificados em 1,5 Mbps, 300 *frames*, resolução cif (352x288), duração de 12s, consistindo em um vídeo de baixa atividade, com pequenos movimentos, (leitura de notícias em um jornal de televisão).
- "soccer.mp4" - codificados em 1,5 Mbps, 300 *frames*, resolução cif (352x288), duração de 12s, consistindo em um vídeo com movimentos rápidos, caracterizado por uma sequência se movendo uniformemente, (partida de futebol).

Ambos os vídeos estão disponíveis em formato .yuv (sendo posteriormente convertidos para MPEG-4) em [43], assim como diversos outros tipos de vídeos.

6.2.3 Métricas de avaliação da qualidade do vídeo

A qualidade do vídeo é de extrema importância quando se analisa serviços de transmissão de vídeos pela internet. Existem duas maneiras de se avaliar a qualidade de um vídeo: através de métricas de qualidade objetivas e métricas de qualidade subjetivas. As métricas de qualidade subjetivas caracterizam-se pela opinião que um observador ou usuário tem sobre a qualidade de um vídeo, basicamente o observador assiste a um vídeo recebido e dá uma nota, consistindo em uma avaliação "subjetiva". Apesar dessa avaliação subjetiva ser de extrema importância, pois no final é a opinião do usuário que conta, esse tipo de avaliação nem sempre é empregada devido ao seu alto custo, pois a informação precisa ser toda analisada, sendo um processo demorado. Muitas métricas de avaliação subjetivas podem ser encontradas em [44]. Uma das métricas subjetivas mais usada é a chamada MOS (*Mean Opinion Score*), sendo a impressão humana da qualidade de um vídeo, dada em uma escala de 1 a 5. O valor 1 representa a pior qualidade e 5 a melhor, como mostrado na Tabela 6.2.

As métricas de qualidade objetivas por sua vez, são métodos matemáticos desenvolvidos para prever automaticamente a qualidade de um vídeo. As métricas objetivas são classificadas comparando-se o vídeo original, com qualidade perfeita e sem distorção, usado como referência, com o vídeo distorcido. A métrica mais utilizada é o cálculo da PSNR (*Peak Signal-to-Noise Ratio*), derivada da SNR. A PSNR é uma medida matemática calculada para cada *frame* de um vídeo, sendo representada por um valor em dB. A PSNR é baseada na MSE (*Mean Square*

Tabela 6.2: Escala de qualidade e degradação da ITU-R [44].

Escala	**Qualidade**	**Degradação**
5	Excelente	Imperceptível
4	Bom	Perceptível, mas sem causar incômodo
3	Razoável	Ligeiramente irritante
2	Ruim	Irritante
1	Péssimo	Muito Irritante

Error) que calcula a diferença entre cada pixel do *frame* original com cada pixel do *frame* distorcido. A equação para o MSE é definida como:

$$MSE = \frac{\sum_{i=1}^{M} \sum_{j=1}^{N} [x(i,j) - y(i,j)]^2}{M \times N}, \tag{6.1}$$

onde $x(i,j)$ e $y(i,j)$ são os posicionamentos dos *pixels* do *frame* original e do *frame* distorcido, respectivamente, na posição (i,j) e $M \times N$ é o tamanho da imagem. E a equação da PSNR é definida como:

$$PSNR = 10 \times \log\left(\frac{MAX^2}{MSE}\right), \tag{6.2}$$

onde MAX é o valor máximo possível de um pixel, ou seja, 255.

Apesar de a PSNR ser ainda muito discutida por não apresentar resultados tão semelhantes aos da análise subjetiva, ela ainda é a mais utilizada dentre as outras métricas pela sua simplicidade.

Uma possível relação da métrica objetiva PSNR para a métrica subjetiva MOS pode ser encontrada na Tabela 6.3. Essa conversão é útil para interpretar a PSNR fazendo uma análise com a avaliação subjetiva e será usado na avaliação dos resultados desse trabalho. Além disso, uma avaliação subjetiva utilizando a opinião de alguns observadores também foi realizada em alguns casos, para verificar se os valores de MOS observados estão de acordo com os da Tabela 6.3. A metodologia de avaliação subjetiva utilizada foi a Double Stimulus Impairment Scale (DSIS) [44], na qual pares de sequências de vídeo são mostradas aos observadores, consistindo em um vídeo de "referência" e um vídeo de "teste". Os observadores avaliam a quantidade de erros na sequência de "teste", considerando a escala de 1 a 5 da Tabela 6.2. Para cada teste foram utilizados 15 observadores.

O software utilizado para calcular as PSNRs para cada *frame*, assim como a PSNR média do vídeo foi o MSU Video Quality Measurement Tools [46]. O software divide o vídeo original e o vídeo distorcido em um determinado número de *frames*, e calcula a PSNR entre

Tabela 6.3: Possível conversão de PSNR em dB para MOS [45].

PSNR	**MOS**
> 37	5 (Excelente)
31 - 37	4 (Bom)
25 - 31	3 (Razoável)
20 - 25	2 (Ruim)
< 20	1(Péssimo)

os *frames*. O valor da PSNR igual a 100 é considerado o valor máximo, quando os *frames* do vídeo original e do vídeo distorcido são iguais.

Porém, os valores de PSNRs calculados nesse trabalho podem não ser valores tão precisos, pois existe uma dificuldade em se calcular as PSNRs de cada *frame* de um vídeo. Quando os *frames* apresentam muito erros, alguns *frames* podem ser perdidos acarretando em uma perda de sincronização entre o vídeo original e o vídeo distorcido, não permitindo que a PSNR seja calculada com precisão. Um possível algoritmo para minimizar esse problema pode ser encontrado em [47].

6.3 Resultados das simulações

Nesta seção serão descritos os resultados das simulações utilizando o Software Modem implementado em Ptolemy, considerando um canal perfeito, apenas com a presença de ruído impulsivo e utilizando o cenário com transmissão de vídeo descrito na Seção 6.2. A qualidade do vídeo baseada na PSNR e na correspondente avaliação subjetiva usando MOS será analisada. Foram realizados dois tipos de simulações. A primeira considerando o modelo de ruído impulsivo adotado (seguindo as distribuições de probabilidade da amplitude, duração e intervalo entre impulsos), e a segunda usando o intervalo entre ruídos com um valor fixo, analisando a qualidade dos vídeos para diferentes valores de *interleaver depths* e mudando a atenuação do sinal. Os resultados serão descritos a seguir.

6.3.1 Simulações com o modelo de ruído impulsivo

Nessa simulação considerou-se o canal perfeito, sem atenuação, apenas com ruído impulsivo, com o sinal com valores de tensão entre -6 V e 6 V e usou-se o modelo de ruído

impulsivo adotado na Seção 2.4.3 e implementado na Seção 5.4. Usando a maior capacidade de correção (R = 16 e D=64) não foi possível que todos os erros causados pelo ruído impulsivo fossem corrigidos. O modelo de ruído impulsivo adotado é caracterizado pela geração de intervalo entre os impulsos muito pequenos (menos de 1 ms) para a maioria dos casos, fazendo com que os ruídos caiam muito próximos, e ainda por conta da propagação de erros do *interleaver depth* (ver Seção 6.3.2), não foi possível que mesmo com a maior capacidade de correção os erros fossem todos corrigidos. Visto que o modelo de ruído impulsivo adotado é não-estacionário, uma quantidade grande de simulações seriam necessárias para que se chegasse a alguma conclusão sobre o modelo, porém o Software Modem ainda não foi otimizado nesse sentido, e as simulações ainda demoram um tempo considerável.

A Figura 6.2 mostra um exemplo de geração de ruídos impulsivo, considerando o sinal de entrada como zero, usando o modelo de ruído impulsivo adotado.

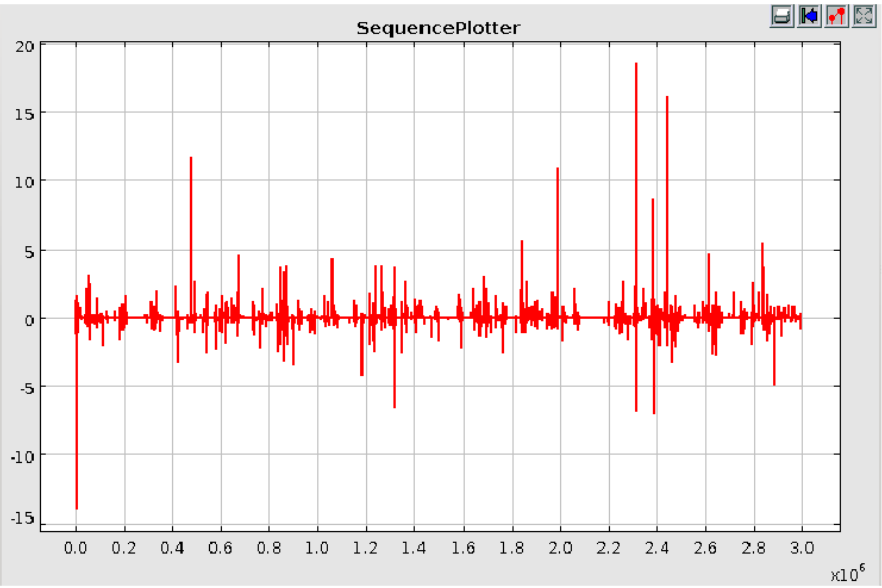


Figura 6.2: Ruído impulsivo. O eixo das abscissas representa as amostras do sinal, e o eixo das ordenadas representa as amplitudes do ruído impulsivo em Volts.

Para esse caso, em um dos testes usando o vídeo “news”, o vídeo recuperado apresentou uma PSNR média de 29,47 dB em uma das simulações. Usando a Tabela 6.3 e fazendo a relação entre PSNR e MOS, o vídeo pode ser caracterizado como um vídeo com qualidade razoável. O efeito do ruído não foi tão agressivo nesse caso possivelmente pelo fato de o sinal estar com um nível de amplitude alto (sinal ADSL sendo transmitido em um canal sem nenhuma atenuação). Considerando uma atenuação de 10 dB no sinal e realizando o mesmo tipo de simulação, para o modelo de ruído impulsivo adotado e a mesma capacidade de correção, não

foi possível nem ao menos reproduzir o vídeo, devido a quantidade de erros ser excessiva.

6.3.2 Simulação com intervalo entre ruídos fixo

Na segunda simulação, foi utilizado primeiramente o canal perfeito, sem atenuação (0 dB), apenas com ruído impulsivo, e com o sinal com valores de tensão entre -6 V e 6 V, aproximadamente. Porém, nesse caso, por motivos de comparação, ou seja, para permitir que fosse possível comparar a qualidade do vídeo usando diferentes valores de *interleaver depths*, o ruído impulsivo foi configurado para ocorrer em intervalos regulares, assim como mencionado por Nedev em [14]. No caso dos testes realizados, foi escolhido o valor do *inter-arrival time* de 3 ms, que mostrou-se um valor razoável para realizar as comparações alterando os *interleavers depths*, no caso do canal considerado perfeito, sem atenuação. As amplitudes e as durações dos impulsos, no entanto, se mantiveram as mesmas do modelo adotado na Seção 2.4.3, seguindo as mesmas distribuições de probabilidade.

Primeiramente, um bom exemplo de como os erros gerados pelo ruído impulsivo se relacionam com a PSNR pode ser visto na Figura 6.3. Nesse caso, o valor do intervalo entre impulsos foi fixado em 100 ms para permitir uma melhor visualização dos impulsos. E o cenário utilizado não considera codificação, para permitir a visualização de como os impulsos afetam o vídeo antes de serem corrigidos. Além disso, como os impulsos no software modem são gerados em amostras, para se fazer a relação com a PSNR, considerou-se que cada símbolo DMT carrega 255 bytes e usando o número de bytes do vídeo, obteve-se o número de símbolos DMT por *frame* do vídeo, resultando em aproximadamente 25 símbolos DMT, o que equivale a aproximadamente 12436 amostras (não se considerou o prefixo cíclico). Através desse número de amostras por *frame* do vídeo, e dos valores de PSNR para cada *frame* do vídeo, assim como das amostras do impulso, foi possível gerar o gráfico da Figura 6.3.

Observa-se pelo gráfico que quando não ocorrem impulsos ou quando as amplitudes dos impulsos são muito baixas, possivelmente menores que as do sinal, o valor da PSNR estabiliza no valor máximo igual a 100. Porém, quando os impulsos ocorrem os valores das PSNRs caem, decorrente dos erros nos vídeos.

Antes da análise dos resultados, é importante ressaltar o impacto do *interleaver* na tentativa de correção de erros. Segundo [14, 48], a probabilidade de erro de *frame* aumenta inicialmente com o aumento do *interleaver depth*. Para exemplificar esse comportamento, a Figura 6.4 (extraída de [14]) mostra como varia a probabilidade de erro de *frame*, em função do aumento do *interleaver depth* para diferentes taxas. Isso ocorre porque os erros são espalhados por mais de uma palavra-código. Se a propagação dos erros é insuficiente para que o FEC possa

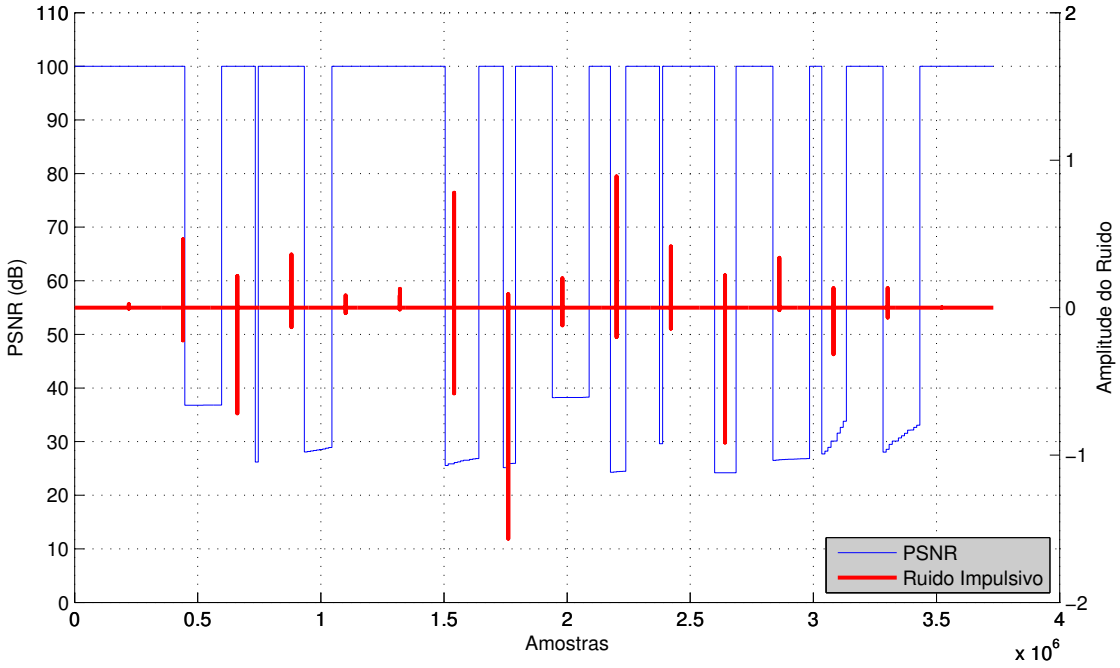


Figura 6.3: PSNR por *frame* de vídeo (cada *frame* equivale a 12436 amostras) e amplitudes do ruído ao longo do tempo.

corrigir, ou seja, o número de erros nas palavras-código ainda é maior do que a capacidade de correção do código RS, então os erros não podem ser corrigidos, e os erros acabam sendo apenas espalhados por mais palavras-código. No entanto, a partir de um valor de corte do *interleaver depth*, a propagação dos erros começa a ser suficiente para que o código do FEC possa corrigir os erros, como pode ser visto na Figura 6.4, quando o valor da probabilidade de erro começa a decair. No entanto, segundo [48], apesar do *interleaver* ter o efeito de multiplicar a taxa de erros de *frames*, ele reduz a probabilidade de erro de bytes.

Para esse cenário, usando o vídeo "news" foram realizadas 5 simulações para cada valor de *interleaver depth* e considerando os casos sem *interleaver* (que representa o caminho *fast path*) e sem codificação. Para cada simulação foram calculadas as PSNRs médias dos vídeos, e a média desses resultados é mostrada na Tabela 6.4, juntamente com o valor correspondente da MOS resultante da Tabela 6.2 e o valor da MOS utilizando a metodologia de avaliação subjetiva DSIS.

Através da tabela, observa-se que as PSNRs para os *interleavers depths* de 2 a 16 apresentam valores na média próximos e os vídeos apresentam a mesma qualidade, com valores um pouco melhores do que nos casos sem *interleaver* e sem codificação. Apenas a partir do

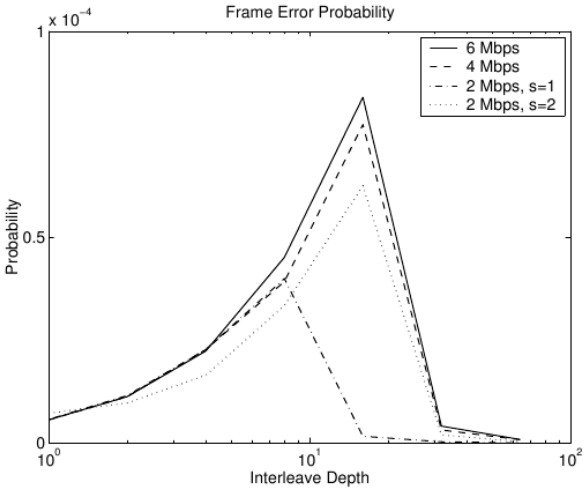


Figura 6.4: Probabilidade de erro de *frame* em função do *interleaver depth* para diferentes taxas [14].

interleaver depth de 32 que a PSNR aumenta e a qualidade do vídeo já é considerada boa. Para o *interleaver depth* de 64 o valor da PSNR é máximo, ou seja, foi possível corrigir todos os erros e recuperar completamente o vídeo.

Esse comportamento pode ser explicado através da análise do impacto do *interleaver* citado em [48]. O *interleaver depth* de corte aqui pode ser considerado o 16, ou seja até o *interleaver depth* de 16 os erros estão apenas sendo espalhados pelas palavras códigos, sem ainda serem corrigidos e as PSNRs estão se mantendo com valores próximos, porém baixos (isso se deve ao fato de que apesar de os erros estarem sendo espalhados, o números de bytes errados total no vídeo é aproximadamente o mesmo, possivelmente apenas alguns bytes estão sendo corrigidos, fazendo com que a PSNR se mantenha com valores próximos, apesar do aumento do *interleaver depth*). A qualidade ainda é considerada razoável nesse caso, e as PSNRs não são tão baixas, pois o sinal é considerado alto, ou seja, o ruído impulsivo não é tão agressivo. A partir do *interleaver depth* de 32 a PSNR aumenta, assim como a qualidade do vídeo melhora, nesse caso a propagação de erros começa a ser suficiente para que a maioria dos erros possam ser corrigidos. Já com D=64, a propagação é suficiente para que todos os erros sejam corrigidos.

Esse efeito pode ser melhor visualizado através do gráfico da Figura 6.5. Utilizou-se uma das simulações para representar o número de *frames* errados em função do *interleaver depth* e também os valores da PSNR para cada *interleaver depth*.

Os resultados mostram que o número de *frames* errados aumenta até D=16, e a partir

Tabela 6.4: Média dos resultados das PSNRs médias do vídeo "news" para diferentes *interleaver depths*, considerando 0 dB de atenuação e espaçamento de 3 ms entre os ruídos.

***Interleaver depth* (D)**	**PSNR média**	**MOS**	**MOS (DSIS)**
64	100	5 (Excelente)	5 (Excelente)
32	35,75	4 (Bom)	4 (Bom)
16	27,25	3 (Razoável)	3 (Razoável)
8	27,30	3 (Razoável)	3 (Razoável)
4	27,22	3 (Razoável)	3 (Razoável)
2	26,05	3 (Razoável)	2 (Ruim)
Sem *Interleaver*	24,57	2 (Ruim)	2 (Ruim)
Sem codificação	24,28	2 (Ruim)	2 (Ruim)

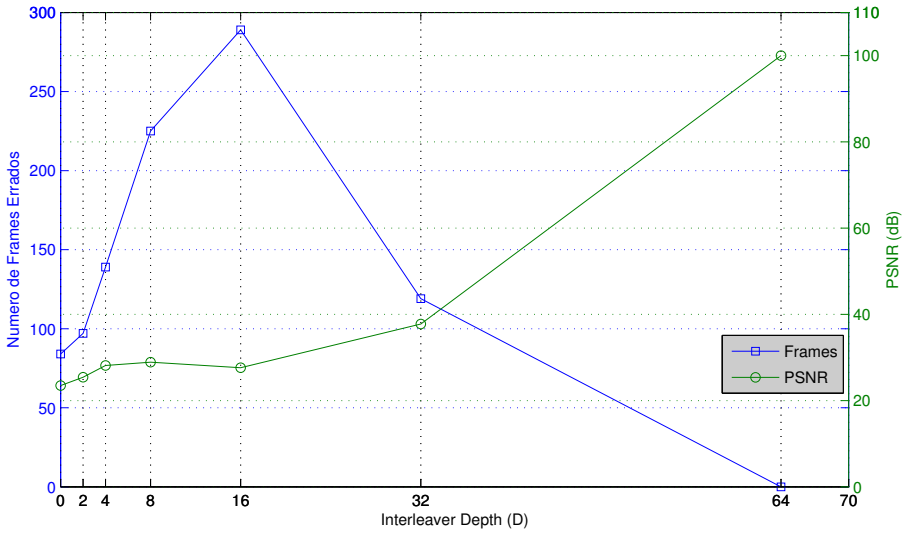


Figura 6.5: Número de *frames* errados e PSNR (dB) em função do *interleaver depth*.

desse valor de corte, o número de *frames* errados decai para D=32 e termina em zero para D=64. Já a PSNR se mantém razoavelmente estável para os *interleavers depths* até o 16 e um pouco abaixo para o caso sem *interleaver* (D=0). A partir do D=32 os valores da PSNR aumentam, mostrando que os *frames* errados agora estão sendo corrigidos.

Olhando para a Tabela 6.4, observa-se também que a diferença entre as PSNRs dos casos sem *interleaver* e sem codificação para os casos com D de 2 a 16 não é considerada grande,

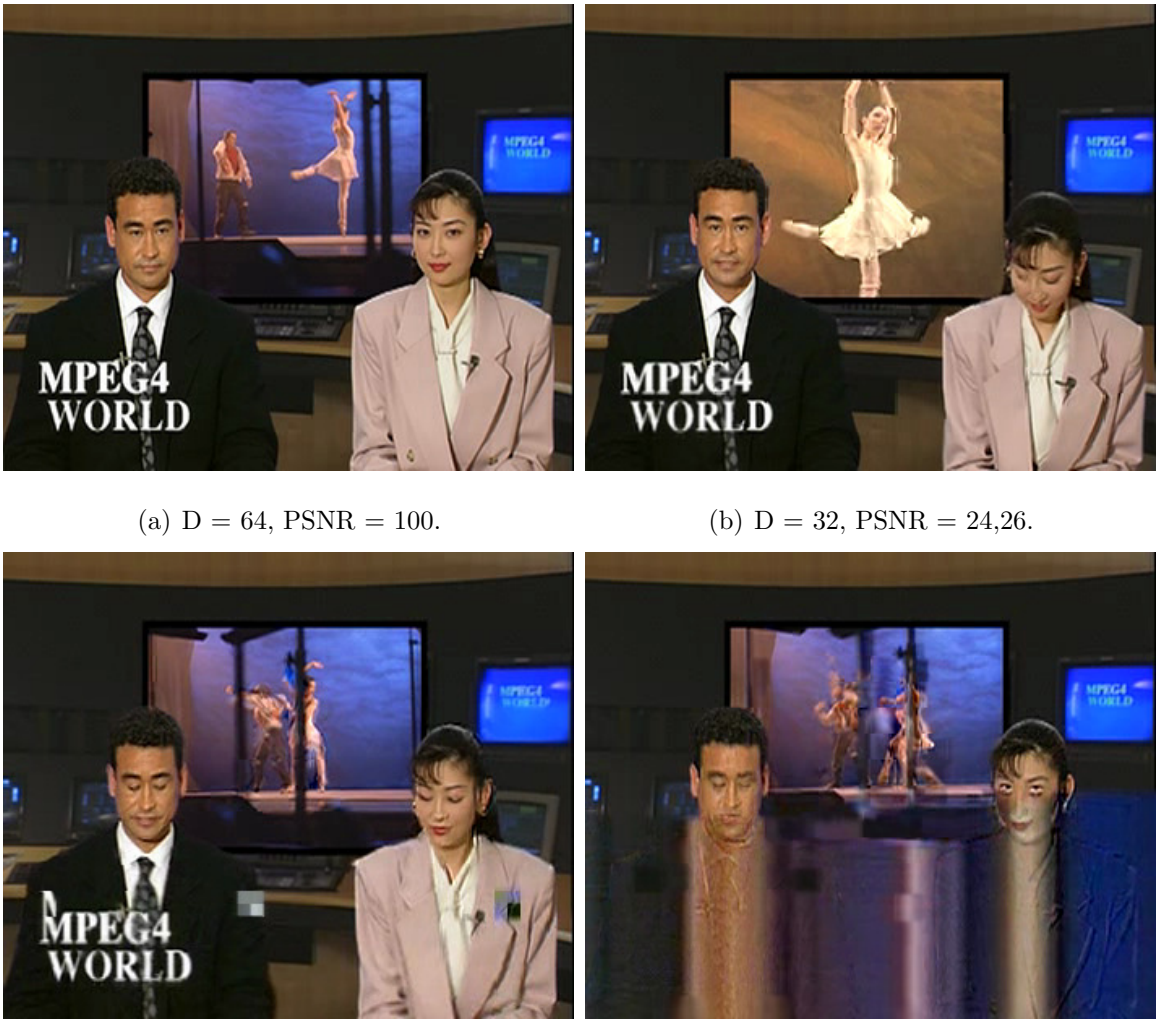
supõe-se que alguns bytes estão sendo corrigidos, porém ainda não o suficiente para garantir uma melhora na qualidade dos vídeos, pode-se considerar que nesse caso o uso de *interleavers* com D até 16 não traz muitas vantagens. O recomendado nesse caso seria utilizar D=32 ou 64, dependendo da aplicação, ou seja, dependendo do nível de *delay* (atraso) permitido. Além disso, observa-se também que os valores da MOS obtidos pela Tabela 6.2 estão condizentes aos valores obtidos com os testes de avaliação subjetiva.

A Figura 6.6 mostra alguns *frames* do vídeo "news" com diferentes quantidades de erros devido a presença do ruído impulsivo considerando uma das simulações. Foram utilizados os vídeos das simulações com as PSNRs médias mais baixas e também o *frame* com a pior PSNR, para alguns valores de *interleaver depth*. As PSNRs foram calculadas para cada *frame* individualmente. A Figura 6.6 (a) mostra um *frame* do vídeo com D=64, no qual nenhum erro foi observado em comparação com o *frame* original e a PSNR é igual ao valor máximo de 100. Na Figura 6.6 (b) com D=32, pequenos erros foram observados, nesse caso foi possível corrigir a maioria dos erros. A Figura 6.6 (c) mostra uma quantidade razoável de erros que começam a afetar a qualidade da imagem e a Figura 6.6 (d) mostra o pior caso, quando nenhuma codificação foi aplicada, a PSNR calculada é baixa e os erros são bastante perceptíveis.

Foram realizadas também 5 simulações usando o vídeo "soccer", caracterizado como um vídeo com bastante movimento. Para cada simulação foram calculadas as PSNRs médias dos vídeos, e a média desses resultados é mostrada na Tabela 6.5, juntamente com o valor correspondente da MOS resultante da Tabela 6.2 e o valor da MOS utilizando a metodologia de avaliação subjetiva DSIS.

Tabela 6.5: Média dos resultados das PSNRs médias do vídeo "soccer" para diferentes *interleaver depths*, considerando 0 dB de atenuação e espaçamento de 3 ms entre os ruídos.

***Interleaver depth* (D)**	**PSNR média**	**MOS**	**MOS (DSIS)**
64	49,44	5 (Excelente)	4 (Bom)
32	31,61	4 (Bom)	4 (Bom)
16	21,60	2 (Ruim)	2 (Ruim)
8	19,85	1 (Péssimo)	2 (Ruim)
4	20,20	2 (Ruim)	2 (Ruim)
2	19,35	1 (Péssimo)	2 (Ruim)
Sem *Interleaver*	19,59	1 (Péssimo)	1 (Péssimo)
Sem codificação	19,11	1 (Péssimo)	1 (Péssimo)



(a) D = 64, PSNR = 100.

(b) D = 32, PSNR = 24,26.

(c) D = 8, PSNR = 19,69.

(d) Sem codificação, PSNR = 13,27.

Figura 6.6: *Frames* do vídeo "news" com diferentes quantidades de erros considerando a presença de ruído impulsivo. PSNRs calculadas para cada *frame* individualmente.

Nesse caso, observa-se que as PSNRs obtiveram o mesmo comportamento do exemplo com o vídeo "news". Os valores das PSNRs considerando os casos sem *interleaver*, sem codificação e com D de 2 a 16 se mantiveram com valores na média próximos. A diferença nas qualidades entre Ruim e Péssimo observadas no valor da MOS, se deve basicamente aos valores estarem muito próximos do limite considerado pela Tabela 6.3. No entando, fazendo a análise subjetiva dos vídeos, observou-se que a qualidade se manteve baixa e praticamente igual para todos os casos. Comparando-se a Tabela 6.5 com a Tabela 6.4, observa-se que os valores absolutos das PSNRs para o vídeo "soccer" se mostraram menores do que os do vídeo "news". Isso se deve provavelmente ao fato do vídeo "soccer" apresentar mais movimento, resultando em mais erros nas imagens. Nesse caso também, apesar de com D=64 se mostrar

o melhor caso, em algumas simulações não foi possível corrigir todos os erros do vídeo.

A Figura 6.7 mostra alguns *frames* do vídeo com diferentes quantidades de erros devido a presença do ruído impulsivo considerando uma das simulações com o vídeo "soccer". A Figura 6.7 (a) mostra um *frame* do vídeo original, apenas para comparação. Na Figura 6.7 (b) com D=64, pequenos erros foram observados, nesse caso, não foi possível que todos os erros fossem corrigidos, como no caso do vídeo "news". A Figura 6.7 (c), com D=32, mostra uma quantidade um pouco maior de erros, se comparado com D=64. A Figura 6.7 (d) mostra uma quantidade razoável de erros que começam a afetar a qualidade da imagem, usando D=16 e a Figura 6.7 (e) mostra o pior caso, quando nenhuma codificação foi aplicada, a PSNR calculada é baixa e os erros são bastante perceptíveis.

Foram consideradas também simulações com o canal com atenuação de -10 dB, com presença de ruído impulsivo e com o sinal com valores de tensão entre -0,6 V e 0,6 V, aproximadamente. Para esse caso, o valor de 3 ms do *inter-arrival time*, não se mostrou suficiente. Devido à redução no nível do sinal, mesmo considerando D=64, o vídeo continha muitos erros, não podendo nem ser recuperado. Portanto, nesse caso, foi escolhido o valor do *inter-arrival time* de 15 ms. Assim como na simulação anterior, as distribuições de probabilidade da amplitude e da duração do ruído se mantiveram as mesmas da modelo adotado na Seção 2.4.3. Não foram realizados os testes de avalição subjetiva para essas simulações, sendo utilizados apenas os valores de MOS provenientes da Tabela 6.2.

Considerando a atenuação no sinal de -10 dB, foram realizadas 5 simulações usando o vídeo "news", na qual para cada simulação foram calculadas as PSNRs médias dos vídeos. A Tabela 6.6 mostra a média dos resultados, juntamente com o valor correspondente da MOS.

Através da Tabela 6.6, observa-se que o comportamento verificado na Tabela 6.4 se repete. Os valores das PSNRs para os *interleaver depths* de 2 a 16 são na média valores próximos, assim como nos casos sem codificação e sem *interleaver*, com os vídeos apresentando a mesma qualidade. A diferença entre as tabelas, é que no caso da Tabela 6.6, os valores absolutos das PSNRs são menores do que no caso da Tabela 6.4. Isso se deve ao fato de que o sinal agora possui amplitudes mais baixas devido à atenuação e por isso o ruído impulsivo se torna bem mais agressivo (mesmo com o espaçamento maior).

O gráfico da Figura 6.8 mostra o número de *frames* errados em função do *interleaver depth* e também os valores da PSNR para cada *interleaver depth*, considerando agora a atenuação de -10 dB.

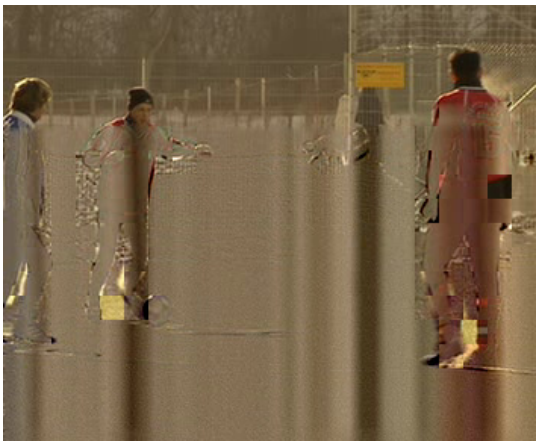
Comparando o gráfico da Figura 6.8 com o gráfico da Figura 6.5 observa-se que o comportamento dos gráficos é semelhante, o valor das PSNRs se mantém estáveis quando ocorre o aumento do número de *frames* errados, até o valor de D=16. A partir desse

(a) Vídeo original.

(b) D = 64, PSNR = 22,51.

(c) D = 32, PSNR = 20,08.

(d) D = 16, PSNR = 16,61.



(e) Sem codificação, PSNR = 13,17.

Figura 6.7: *Frames* do vídeo "soccer" com diferentes quantidades de erros considerando a presença de ruído impulsivo. PSNRs calculadas para cada *frame* individualmente.

Tabela 6.6: Média dos resultados das PSNRs médias do vídeo "news" para diferentes *interleaver depths*, considerando -10 dB de atenuação e espaçamento de 15 ms entre os ruídos.

***Interleaver depth* (D)**	**PSNR média**	**MOS**
64	100	5 (Excelente)
32	33,57	4 (Bom)
16	22,13	2 (Ruim)
8	22,05	2 (Ruim)
4	22,10	2 (Ruim)
2	21,03	2 (Ruim)
Sem *Interleaver*	22,16	2 (Ruim)
Sem codificação	22,05	2 (Ruim)

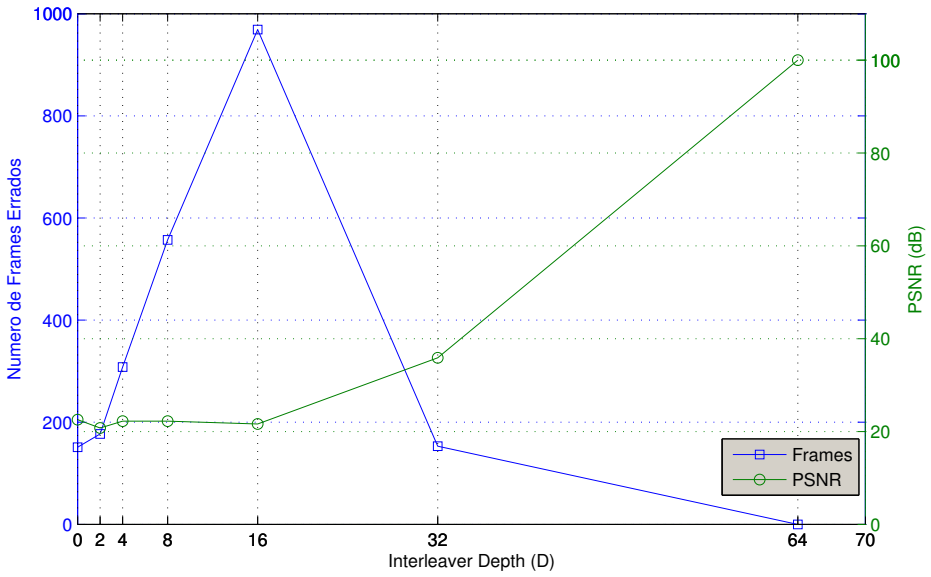
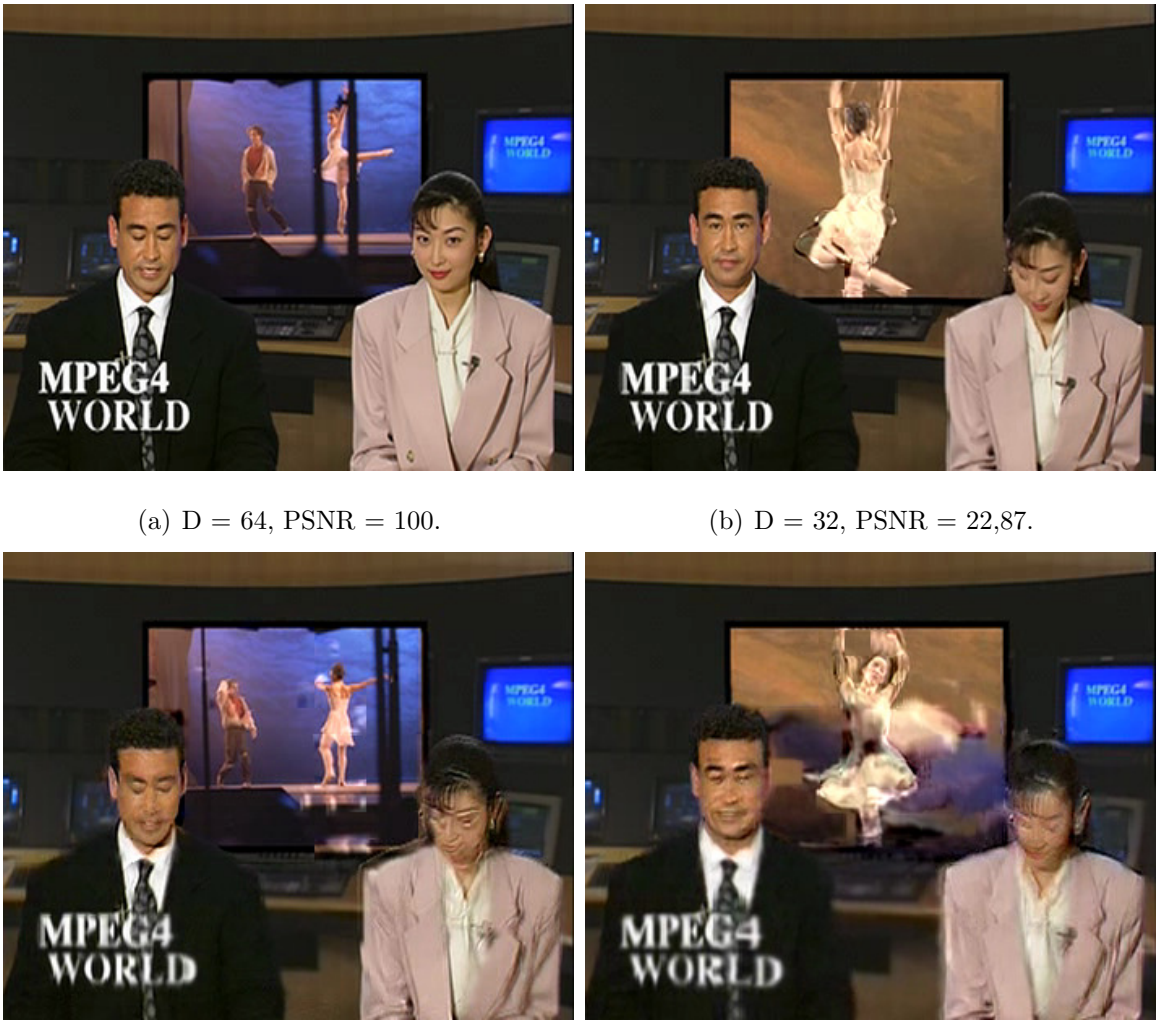


Figura 6.8: Número de *frames* errados e PSNR (dB) em função do *interleaver depth*, considerando atenuação de -10 dB.

valor o número de *frames* errados decai, mostrando que os *frames* estão sendo corrigidos, e consequentemente o valor da PSNR aumenta. No entanto, o gráfico da Figura 6.8 apresenta o número de *frames* errados (para um dado *interleaver depth*) maior do que no gráfico da Figura 6.5.

A Figura 6.9 mostra alguns *frames* de uma das simulações usando o vídeo "news" com diferentes quantidades de erros devido a presença do ruído impulsivo, considerando a atenuação de -10 dB do sinal. Assim como na Figura 6.6, a Figura 6.9 (a) mostra um *frame* do vídeo com D=64, no qual nenhum erro foi observado em comparação com o *frame* original e a PSNR é igual ao valor máximo de 100. Na Figura 6.9 (b) com D=32, poucos erros foram observados, já na Figura 6.9 (c) a quantidade de erros começa a afetar consideravelmente a qualidade da imagem usando D=16, e a Figura 6.9 (d) mostra o pior caso, quando não há codificação e os erros são bastante perceptíveis.



(a) D = 64, PSNR = 100.

(b) D = 32, PSNR = 22,87.

(c) D = 16, PSNR = 19,51.

(d) Sem codificação, PSNR = 16,94.

Figura 6.9: *Frames* do vídeo "news" com diferentes quantidades de erros considerando a presença de ruído impulsivo, para o caso com atenuação de -10 dB. PSNRs calculadas para cada *frame* individualmente.

Foram realizadas também 5 simulações para o vídeo "soccer" considerando a atenuação

no sinal de -10 dB. A Tabela 6.7 mostra a média dos resultados do cálculo das PSNRs médias do vídeo, juntamente com o valor correspondente da MOS.

Tabela 6.7: Média dos resultados das PSNRs médias do vídeo "soccer" para diferentes *interleaver depths*, considerando 10 dB de atenuação e espaçamento de 15 ms entre os ruídos.

***Interleaver depth* (D)**	**PSNR média**	**MOS**
64	44,90	5 (Excelente)
32	25,30	3 (Razoável)
16	17,20	1 (Péssimo)
8	17,54	1 (Péssimo)
4	18,80	1 (Péssimo)
2	17,64	1 (Péssimo)
Sem *Interleaver*	17,75	1 (Péssimo)
Sem codificação	17,72	1 (Péssimo)

Através da Tabela 6.7 observa-se que as PSNRs obtiveram o mesmo comportamento da Tabela 6.5. No caso da Tabela 6.7 os valores absolutos das PSNRs são menores do que na Tabela 6.5, apresentando também pior qualidade, devido as amplitudes do sinal serem mais baixas. Esse foi considerado o pior caso, com os menores valores de PSNRs. Nesse caso também em algumas simulações não foi possível corrigir todos os erros do vídeo considerando D=64, como nos casos dos vídeos "news".

A Figura 6.10 mostra alguns *frames* de uma das simulações usando o vídeo "soccer" com diferentes quantidades de erros devido a presença do ruído impulsivo, considerando a atenuação de -10 dB do sinal. Assim como na Figura 6.7, a Figura 6.10 (a) mostra um *frame* do vídeo original, apenas para comparação. Na Figura 6.10 (b) com D=64, pequenos erros foram observados, nesse caso, não foi possível que todos os erros fossem corrigidos, assim como no caso do vídeo "soccer" sem atenuação. A Figura 6.10 (c), com D=32, mostra uma quantidade um pouco maior de erros, se comparado com D=64. A Figura 6.10 (d) mostra uma quantidade razoável de erros que começam a afetar a qualidade da imagem, usando D=16 e a Figura 6.10 (e) mostra o pior caso, quando nenhuma codificação foi aplicada, a PSNR calculada é baixa e os erros são bastante perceptíveis.

(a) Vídeo original.

(b) D = 64, PSNR = 25,65.

(c) D = 32, PSNR = 22,77.

(d) D = 16, PSNR = 18,58.

(e) Sem codificação, PSNR = 13,55.

Figura 6.10: *Frames* do vídeo "soccer" com diferentes quantidades de erros considerando a presença de ruído impulsivo, para o caso com atenuação de -10 dB. PSNRs calculadas para cada *frame* individualmente.

Capítulo 7

Conclusões

Este trabalho apresentou uma implementação em software da codificação de canal utilizada no padrão ADSL. Os sistemas ADSL permitem que velocidades de transmissão de dados altas possam ser transmitidas através dos pares trançados já utilizados na telefonia convencional. A dissertação incluiu um breve histórico e a descrição dos fundamentos dos sistemas DSL, em especial o ADSL, incluindo os tipos de interferência que atingem esse tipo de sistema. Destaque especial foi dado ao ruído impulsivo, levado em conta nos resultados deste trabalho. A teoria de codificação de canal também foi apresentada, para facilitar o entendimento da codificação de canal aplicada especificamente aos sistemas ADSL. É importante salientar que sistemas mais modernos que o ADSL já existem, incluindo uma atualização do próprio ADSL, chamada de ADSL2+. No entanto, o ADSL mostrou-se mais adequado para a implementação de um modem em software.

A dissertação descreveu o ambiente de desenvolvimento, o Ptolemy II, e os blocos implementados do Software Modem ADSL, destacando, obviamente, os blocos de codificação, que são o foco deste trabalho e foram totalmente implementados pela autora. A implementação do modelo de ruído impulsivo também foi realizada pela autora e é descrita no trabalho.

Para demonstrar que a codificação de canal implementada segue o padrão ADSL, testes foram realizados utilizando o equipamento para análises de sistemas DSL TraceSpan. Nestes testes, foi possível constatar que as sequências de bytes decodificadas utilizando o TraceSpan e os blocos de codificação de canal implementados são idênticas.

Como exemplo de aplicação do Software Modem ADSL e para demonstrar o efeito da codificação de canal no combate ao ruído, foi incluído um estudo de caso sobre os efeitos do ruído impulsivo na transmissão de vídeo.

Uma análise do impacto do *interleaver* na correção dos erros causados pelo ruído

impulsivo foi realizada. Dois tipos de vídeo MPEG-4 foram utilizados nas simulações, denominados " news" e "soccer". O primeiro caracterizado por um vídeo com pouco movimento e o segundo caracterizado por movimentos rápido. A qualidade dos vídeos baseada na PSNR e no valor correspondente da MOS foi discutida. Os valores do *interleaver depth* foram variados (2, 4, 8, 16, 32 e 64) e os casos sem *interleaver* e sem codificação foram considerados. A atenuação do sinal também foi variada, considerando o caso sem atenuação e com atenuação de -10 dB.

Para permitir as comparações entre os *interleaver depths*, os intervalos entre ruídos foram configurados com um valor fixo. Para ambos os vídeos observou-se que com os *interleaver depth* de 2 a 16, os valores das PSNRs ficaram próximos, apresentando a mesma qualidade para a maioria dos casos. Avaliando-se o número de *frames* errados resultantes de cada *interleaver depth*, constatou-se o efeito da propagação de erros de *frames* do *interleaver*, resultando no aumento do número de *frames* errados até o valor de D = 16. A partir do *interleaver depth* de 32 os valores das PSNRs aumentaram, assim como a qualidade, devido a propagação de erros começar a ser suficiente para que a maioria dos erros fossem corrigidos. Para o *interleaver depth* de 64, no caso do vídeo "news" foi possível que todos os erros fossem corrigidos, no entanto, no caso do vídeo "soccer", apesar de com D=64 ser considerado o melhor caso, em algumas simulações não foi possível corrigir todos os erros.

Para as simulações considerando o sinal sem atenuação, ambos os vídeos apresentaram valores de PSNRs maiores e qualidades melhores se comparados com o caso com atenuação de -10 dB. Isso se deve ao fato de que no caso sem atenuação o nível do sinal é considerado alto, fazendo com que o ruído não seja tão agressivo.

O vídeo "soccer" apresentou valores de PSNRs menores se comparados com o vídeo "news", por ser caracterizado com um vídeo com bastante movimento, resultando em uma quantidade maior de erros nas imagens.

É importante ressaltar que a implementação da codificação de canal do Software modem ADSL faz parte de um projeto de construção de um modem ADSL completo em software, trazendo diversas vantagens. Particularmente, a implementação da codificação de canal do ADSL traz a possibilidade de realizar simulações em *showtime* considerando a transmissão de qualquer tipo de arquivo, permitindo a análise de dados e também a implementação de outros tipos de canais ou diferentes cenários.

7.1 Trabalhos Futuros

Como sugestão para trabalhos futuros, pode-se destacar:

- Implementação dos blocos de codificação de sistemas DSL mais modernos, como o ADSL2+ e o VDSL2;
- Otimização do Software Modem ADSL para reduzir o tempo de simulação. Isso permitiria um estudo mais aprofundado dos efeitos do ruído impulsivo na transmissão de vídeo;
- Implementação e teste de potenciais esquemas de codificação para o novo padrão de sistemas DSL em desenvolvimento, chamado atualmente de 4GBB (quarta geração de sistemas banda larga).

Publicações do Autor no Período

Artigos de Conferência

- **"On the Effectiveness of Dynamic Spectrum Management Algorithms in xDSL Networks"**. Fernanda Smith, Marcio Monteiro, Francisco Müller, Boris Dortschy, Aldebaro Klautau e Evaldo Pelaes. Em *International Telecommunications Symposium* - ITS 2010.

Referências Bibliográficas

[1] T. Starr, J. M. Cioffi, and P. J. Silverman, *Understanding Digital Subscriber Line Technology*. Prentice-Hall, 1999.

[2] T. Starr, M. Sorbara, J. M. Cioffi, and P. J. Silverman, *DSL Advances*. Prentice-Hall, 2003.

[3] ITU-T, "Asymmetric Digital Subscriber Line (ADSL) transceivers," June 1999.

[4] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng, "Overview of the Ptolemy Project," University of California, Berkeley, Tech. Rep., July 2, 2003.

[5] ITU-T, "Very high speed digital subscriber line transceivers 2 (VDSL2)," February 2006.

[6] "ADSL transceiver design - UT Austin ADSL2 simulator." [Online]. Available: http://users.ece.utexas.edu/~bevans/projects/adsl/index.html

[7] Conniq.com, "xDSL comparison - http://www.conniq.com/xDSL_comparison.htm, último acesso em 28/02/2011."

[8] P. Golden, H. Dedieu, and K. Jacobsen, *Fundamentals of DSL Technology*. Auerbach Publications, Taylor & Francis Group, 2006.

[9] ANSI, "T1.413 - Network to customer installation interface - Asymmetric Digital Subscriber Lines (ADSL) Metallic Interface," 2004.

[10] J. M. Cioffi, "Stanford class notes – http://www.stanford.edu/group/cioffi/," 2010.

[11] N. B. N. Group, "xDSL modulation technique: methods of achieving spectrum-efficient modulation of high quality tranmsission," May 2001.

[12] J. G. Proakis, *Digital Communications*, 4th ed. McGraw-Hill, 2001.

[13] J. Cook, R. Kirkby, M. Booth, K. Foster, D. Clarke, and G. Young, "The noise and crosstalk environment for ADSL and VDSL systems," *IEEE Commu. Mag.*, vol. 37, no. 5, pp. 73–78, May 1999.

[14] N. H. Nedev, "Analysis of the impact of impulse noise in digital subscriber line systems," Ph.D. dissertation, The University of Edinburgh, Mar. 2003.

[15] W. Henkel, T. Kessler, and H. Y. Chung, "Coded 64-CAP ADSL in an impulse-noise environment-modeling of impulse noise and first simulation results," *IEEE Journal, On Selected Areas in Communications*, vol. 13, December 1995.

[16] S. McLaughlin, W. Henkel, R. Kirkby, and T. Kessler, "Text for realistic impulsive noise model," *Submission to ETSI WG TM6, TD20, 011T20*, February 2001.

[17] I. Mann, S. McLaughlin, W. Henkel, R. Kirkby, and T. Kessler, "Impulse Generation With Appropriate Amplitude, Length, Inter-Arrival, and Spectral Characteristics," *IEEE J. Select. Areas Commun.*, vol. 20, no. 5, pp. 901–912, June 2002.

[18] W. Henkel and T. Kessler, "A wideband impulsive noise survey in the german telephone network: statistical description and modelling," *AEU*, vol. 48, pp. 277–288, November/December 1994.

[19] D. B. Levey and S. McLaughlin, "The statistical nature of impulse noise interarrival times in digital subscriber loop systems," *Signal Processing*, vol. 82, pp. 329–351, 2002.

[20] R. J. A. Tough and K. D. Ward, "The correlation properties of gamma and other non-gaussian processes generated by memoryless nonlinear transformation," *J. Physics D: Applied Physics*, vol. 32, pp. 3075–3084, Dec. 1999.

[21] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 28, 1948.

[22] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 26, 1950.

[23] M. J. E. Galoy, "Notes on digital coding," *Proceedings of the I. R. E. (I. E. E. E.)*, vol. 37, 1949.

[24] J. C. Moreira and P. G. Farrell, *Essentials of Error-Control Coding*. John Wiley e Sons, Ltd., 2006.

[25] S. Haykin, *Sistemas de Comunicação: Analógicos e Digitais*, 4th ed. Bookman, 2004.

[26] S. Lin and D. C. Jr., *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 1983.

[27] R. H. Morelos-Zaragoza, *The Art of Error correcting Coding*. John Wiley e Sons, Ltd., 2002.

[28] D. G. Hoffman, D. A. Leonard, C. C. Lindner, K. T. Phelps, C. A. Hodger, and J. R. Wall, *Coding Theory: The Essentials*. Marcel Dekker, Inc., 1991.

[29] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, pp. 68–79, March 1960.

[30] I. Reed and G. Solomom, "Polynomial codes over certain finite fields," *SIAM Journal of Applied Math*, vol. 8, pp. 300–304, 1960.

[31] B. Sklar, *Digital communications: Fundamentals and applications*, 2nd ed. Prentice Hall, 2001.

[32] E. R. Berlekamp, "On decoding binary bose-chaudhuri-hocquenghem codes," *IEEE Trans. Inf. Theory*, vol. IT-11, pp. 577–580, October 1965.

[33] J. L. Massey, "Step-by-step decoding of the bose-chaudhuri-hocquenghem codes," *IEEE Trans. Inf. Theory*, vol. IT-11, pp. 580–585, October 1965.

[34] M. Purser, *Introduction to Error-Correcting Codes*. Artech House, 1995.

[35] J. Turnbull and S. Garret, *Broadband Applications and the Digital Home*. The Institution of Engineering and Technology, 2003.

[36] J. A. C. Bingham, *ADSL, VDSL, and Multicarrier Modulation.* John Wiley & Sons, Inc., 2000.

[37] Y. Okunev and Y. Goldstein, "Convolution interleaver and deinterleaver for systems with error correction encoding," U.S. Patent 2003/0 091 109, May 15, 2003.

[38] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng, "Heterogeneous concurrent modeling and design in Java (Volume 3: Ptolemy II domains)," University of California, Berkeley, Tech. Rep., April 1, 2008.

[39] ——, "Heterogeneous concurrent modeling and design in Java (Volume 2: Ptolemy II software architecture)," University of California, Berkeley, Tech. Rep., July 15, 2005.

[40] TraceSpan, "DSL Xpert User's Guide," 2004. [Online]. Available: http://www.tracespan.com/

[41] R. Y. Wang, "ADSL power spectrum density calculation," Texas Instruments, Tech. Rep., September, 2003.

[42] T. Ebrahimi and F. Pereira, *The MPEG-4 Book*. Prentice Hall, 2002.

[43] "Xiph.org Test Media," http://media.xiph.org/video/derf/, último acesso em 12/2010.

[44] ITU-R BT.500, "Methodology for the subjective assessment of the quality of television pictures," 2000.

[45] J. R. Ohm, *Bildsignalverarbeitung fuer multimedia-systeme*. Skript, 1999.

[46] D. Vatolin, A. Moskvin, and O. Petrov, "Msu video quality measurement tool." [Online]. Available: http://compression.ru/video/quality_measure/video_measurement_tool_en.html

[47] G. G. B. Santos, "Video stream optimization in ADSL arquitecture," Master's thesis, Universidade Federal de Pernambuco, 2007.

[48] N. H. Nedev, S. McLaughlin, and D. I. Laurenson, "Estimating errors in xDSL due to impulse noise," *Int. Zurich Seminar on Communications (IZS)*, 2004.